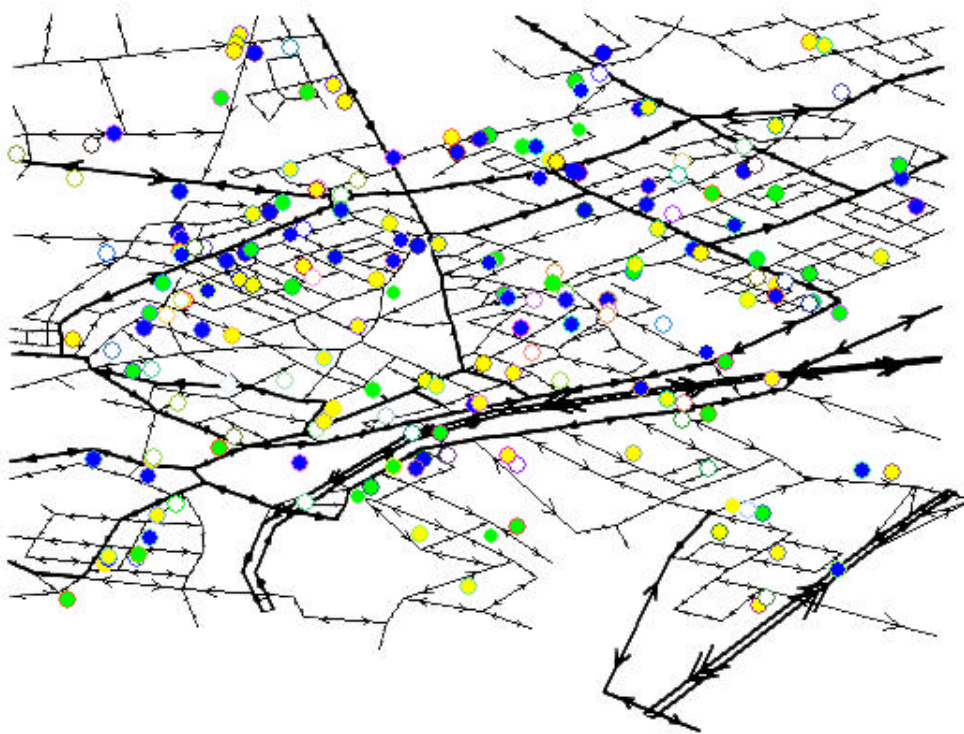


Dynamic vehicle routing using Ant Based Control



Master's thesis of Ronald Kroon

Delft University of Technology
Faculty of Information Technology and Systems
May 2002

Graduation committee:

Dr. drs. L.J.M Rothkrantz

Prof. dr. ir. E.J.H. Kerckhoffs

Prof. dr. H. Koppelaar (chairman)

Kroon, Ronald (R.Kroon@twi.tudelft.nl)

Master's thesis, May 2002

Dynamic vehicle routing using Ant Based Control

Delft University of Technology, The Netherlands
Faculty of Information Technology and Systems
Knowledge Based Systems Group

Keywords: Dynamic routing system, Navigation system, Intelligent agents, Ant Based Control, Distributed routing, Personal route guidance, Traffic simulation.

© Copyright 2002, Ronald Kroon

Acknowledgements

First of all I would like to thank Leon Rothkrantz for his guidance, advice and support throughout the entire project. I feel I have really been lucky to be working with someone like him. It has been amazing how somebody can have so much knowledge of, and interest in my project, while working on so many projects concurrently. Our regular sessions were both enjoyable and inspiring.

Secondly, I would like to thank Patrick Ehlert and the other staff and students of the department Knowledge Based Systems for their help with programming problems and other computer related issues.

I also want to thank my friends Jan Paul van Waveren, Edward van Bilderbeek and Karin de Boer, how made most of the time at the university very enjoyable.

And last but not least, I would like to thank my family and especially Mariëlle Spits for their moral support and more.

Ronald Kroon
Hoofddorp, May 2002

Summary

Static route planners are widely available. But real-time navigation systems that provide optimal routes based on dynamic traffic information on highways and in cities do not yet exist. However, a great profit could be taken from such a system, looking at the amount of daily congestions. On highways, road sensors are capable of gathering a great deal of information about the traffic flow. But in cities such data is not available.

In this thesis we have explored the use of Ant Based Control for dynamic vehicle routing in a city. This approach is based on the behaviour of ants in nature. Intelligent software agents imitate some of the behaviours of the natural ants. Two improvements have been made to earlier versions of the Ant Based Control algorithm. Furthermore we have looked at the possibilities for collecting real-time traffic information from individual vehicles. We have chosen for an approach, which uses GPS and GMS/GPRS for positioning and communication to a central system.

These two components are combined in a routing system that provides car drivers with the shortest route from A to B. Before this routing system can be implemented in the real world, it has to be thoroughly tested. For this purpose a simulation environment has been created, in which traffic in a city can be simulated. The vehicles in the simulation are placed in a realistic situation where they have to deal with matters like traffic lights and precedence rules and roundabouts, and driving on one lane, multi-lane or one-way roads. They provide dynamic data to the routing system by sending their position at regular times. This information is translated by the routing system in travel time estimates. With the aid of these travel times new shortest routes are computed. These routes are sent to the vehicles for navigation. The routing system has proven to be effective in finding good paths, avoiding congestions and roadblocks. The routing system allows for future extensions with even wider applications.

Contents

ACKNOWLEDGEMENTS	I
SUMMARY	III
CHAPTER 1: INTRODUCTION.....	1
1.1 PROBLEM SETTING.....	1
1.2 DYNAMIC VEHICLE ROUTING SYSTEM	2
1.2.1 City traffic.....	2
1.2.2 Control centre	3
1.2.3 Timetable updating system.....	3
1.2.4 Route finding system	3
1.2.5 ABC-algorithm	3
1.3 PROJECT DESCRIPTION.....	4
1.4 REPORT STRUCTURE	4
CHAPTER 2: THEORETIC BACKGROUND.....	5
2.1 ROUTING ALGORITHMS	5
2.1.1 Dijkstra's algorithm.....	5
2.1.2 Ant Based Control.....	6
2.2 NAVIGATION SYSTEMS.....	9
2.2.1 Travelstar.....	9
2.2.2 Tegarons Scout	10
2.2.3 PROMISE	11
2.2.4 Smart Road project.....	11
CHAPTER 3: DESIGN.....	12
3.1 GLOBAL DESIGN	12
3.1.1 Distributed routing system.....	15
3.1.2 Vehicle connection	15
3.2 SYSTEM DESIGN	16
3.3 CITY PROGRAM.....	17
3.3.1 Vehicle movement	19
3.3.2 Intersection handling	22
3.3.3 Control centre	25
3.4 ROUTING SYSTEM	25
3.4.1 Timetable updating system.....	27
3.4.2 Route finding system	31
3.4.3 Distributed components.....	34
CHAPTER 4: IMPLEMENTATION.....	35
4.1 INTRODUCTION.....	35
4.2 TRAFFIC SIMULATION.....	36
4.2.1 Loading a traffic network	37
4.2.2 Changing the traffic network	41
4.2.3 Saving the traffic network.....	41
4.2.4 Automatic simulation preparation.....	42
4.3 ROUTING SYSTEM	45
4.3.1 Loading the agent network	46
4.3.2 Automatic simulation preparation.....	47
4.3.3 Saving routing tables	47
4.4 RUNNING A SIMULATION.....	47

4.4.1	<i>Manual simulation preparation.....</i>	48
4.4.2	<i>Running the simulation.....</i>	50
4.4.3	<i>Evaluating the simulation.....</i>	56
4.5	SPECIAL FUNCTIONALITY	57
4.5.1	<i>Integer queue.....</i>	57
4.5.2	<i>Integer ring.....</i>	57
CHAPTER 5: EXPLOITING THE APPLICATIONS		59
5.1	CITY PROGRAM.....	59
5.1.1	<i>Possibilities</i>	59
5.1.2	<i>Limitations</i>	62
5.2	ROUTING SYSTEM	63
5.2.1	<i>Distributed Routing system</i>	63
5.2.2	<i>Performance issues</i>	64
CHAPTER 6: EXPERIMENTS AND RESULTS		66
6.1	EXPERIMENT 1: ADAPTATION SPEED	66
6.1.1	<i>Goal.....</i>	66
6.1.2	<i>Environment and settings</i>	66
6.1.3	<i>Expectations</i>	67
6.1.4	<i>Results</i>	68
6.2	EXPERIMENT 2: EFFECTIVENESS	71
6.2.1	<i>Goal.....</i>	71
6.2.2	<i>Environment and settings</i>	71
6.2.3	<i>Expectations</i>	73
6.2.4	<i>Results</i>	73
6.3	EXPERIMENT 3: STABILITY	76
6.3.1	<i>Goal.....</i>	76
6.3.2	<i>Environment and settings</i>	76
6.3.3	<i>Expectations</i>	78
6.3.4	<i>Results</i>	78
CHAPTER 7: CONCLUSIONS AND RECOMMENDATIONS		81
7.1	CONCLUSIONS.....	81
7.2	RECOMMENDATIONS	82
7.2.1	<i>Simulation environment</i>	82
7.2.2	<i>Timetable updating system</i>	82
7.2.3	<i>Routing algorithm.....</i>	83
APPENDIX A: BIBLIOGRAPHY		84
APPENDIX B: EXPERIMENTAL RESULTS		86
B.1	EXPERIMENT 1.....	86
B.2	EXPERIMENT 2.....	94
B.3	EXPERIMENT 3.....	95
APPENDIX C: USER MANUAL		107
C.1	THE BASIC STEPS TO OPERATE THE SYSTEM	107
C.2	FEATURES OF THE CITY PROGRAM	109
C.3	FEATURES OF THE ROUTING SYSTEM PROGRAM	117

Chapter 1: Introduction

1.1 Problem setting

Car traffic is getting busier and busier each year. Everyone is familiar with traffic congestion on highways and in the city. And everyone will admit that it is a problem that affects us both economically as well as mentally. It is very frustrating to end up in a traffic jam just when you were in a hurry. But it is even worse that a lot of people know in advance that every day they need twice as much time as the maximum allowed speed would make possible. Furthermore finding your way in an unknown city can be very difficult, even with a map, especially when you are alone.

One of the reasons for congestions is that road capacity is not optimally used. Route information along the road is focussed on “shortest distance”. This information is static. Dynamic route information could exploit the network capacity in an optimal way.

Some service providers, such as ANWB, can provide routes based on static information and additionally they can inform you about possible congestions. Figure 1 was taken from [ANWB]. This information about congestions on highways is available via Internet, SMS, WAP and i-mode. These separate services are however not combined to provide the shortest routes in time, from the available data.

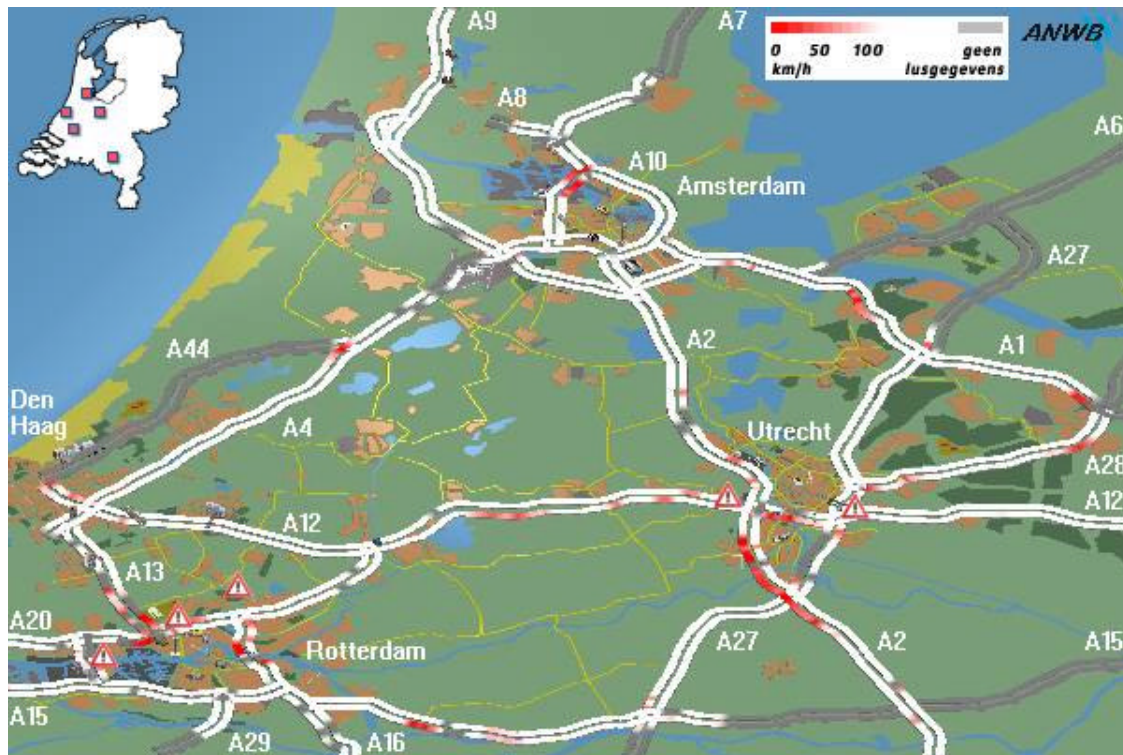


Figure 1: Congestions on highways marked with red

Navigation systems, like VDO Dayton, can display the route to be followed on a display in a car. The user only has to provide his desired destination. The latest versions also use congestion information to avoid trouble spots. In case of an accident or a blocked road this can be an optimal solution. But in case of congestion not all cars can avoid or have to avoid the road. Congestion very is dynamic. Finding the shortest route requires a dynamical approach using the available road capacity.

At this moment MoniCa is a system that collects and provides dynamic route information for car drivers. This information is presented at DRIPs (Dynamic Route Information Panels) along the way. The goal of this system is to route the whole cohort of drivers in an optimal way.

This thesis addresses traffic problems in a *city*. It describes a routing system able to guide *individual* car drivers through a city. Together with that, it deals with a simulation environment that enables experimenting with the routing system in a computer environment. The routing system should help users to find any address in the city. It could also be used to find a free parking place in the centre or near some other address. And more important, it should provide the route with the shortest travel time. It avoids congested roads when other roads are faster. It provides short cuts for the user that didn't know the city at all.

1.2 Dynamic vehicle routing system

To provide up-to-date, dynamic route information it is necessary to have access to a database of traffic congestion and up-to-date travel times. Information from MoniCa is not publicly available. Some information is available via radio broadcast and by phone, but this information is insufficient or not freely available. We came up with the idea that travellers on the route should provide the necessary information themselves. At regular times they send their position to a central database. From that information the time it takes to travel from A to B can be computed. So at the start we have only static route information to provide the shortest routes, but once travellers are on their way, we get recent travelling data. The next step is to compute the shortest route from A to B based on the existing dynamic route information. We have to realise that all the time the optimal route has to be computed, as soon as the travel times are refreshed. We have chosen the ABC-algorithm for dynamic routing. After developing our system, we have to test it. Before it can be implemented in a real world application, we want to make sure that our system works as expected. So it was necessary to design and implement a test bed, a simulation of the traffic in a city.

An application called City consists of both the simulation environment for the traffic and the Control centre, where parameter settings can be changed. The traffic in the simulation communicates with the Routing system. This way we simulate the communication of real vehicles with the Routing system. The Routing system consists of two subsystems. The first subsystem is the Timetable updating system. This subsystem receives information about the traffic network in the city. This information is provided by the vehicles driving through the city. So our proposed system is self-contained. There is no use of external databases for dynamic information. Usually those external databases are not available for public domain use. The second subsystem is the Route finding system. This system calculates the shortest routes for all the users on the basis of the information in the table with the travel times. The used algorithm (Ant Based Control) finds its origin in mechanisms of natural life. This algorithm is used in a lot of traffic routing applications [Dorigo 1997], [Di Caro 1997], [Di Caro 1998], [Schoonderwoerd 1997], [Van der Put 1999], [Knibbe 1999]. A description of the main components follows in the next sections.

1.2.1 City traffic

This is an environment that visualises a traffic network. In this network vehicles are moving from their starting point to their destination. For the route they follow they can depend on a fixed standard route as well as on a dynamic route provided by the Routing system. On their way the drivers have to deal with realistic traffic features like traffic lights and precedence rules. The vehicles must be enabled to communicate with the Routing system for two purposes. The first purpose is to request the route they have to follow. The other purpose is to provide the Routing system with feedback about the load of the network.

1.2.2 Control centre

The Control centre is part of the City program. Its function is to enable varying of parameters. This concerns both the simulation of the traffic network as well as the Routing system. It allows for setting the duration of a simulation, and the load of the traffic network. But it also allows for fine-tuning of parameters used by the routing algorithm.

1.2.3 Timetable updating system

The Timetable updating system can receive its information about the traffic network in the city from different sources. This can for example be directly from sensors in the road-surface. Such sensors can count vehicles and measure the speed of the vehicles. That information can be used by the Timetable updating system to compute the time it takes to cover a part of the road. Another source can be the traffic information services. They can inform the system about congestion, diversions of the road, roadblocks and perhaps open bridges. And finally the vehicles themselves can provide the system with information about the path they followed and the time it took them to cover it. The GPS-technology can be used to locate a vehicle with a precision of a few meters. That position can be communicated to the system along with the covered route.

In our model the crossings are the marking points. The traffic flow between the marking points is computed with the information from the vehicles. This way the travel time for every route from any crossing to any other crossing in the city can be computed by just adding up all the travel times between the marking points.

1.2.4 Route finding system

The Route finding system receives requests for routes from individual motorists. For each motorist the shortest route in time will be calculated and send back to the motorist. This information depends on the time and the location of the vehicle. It consists of an instruction at every crossing where to go next. The route information is updated at regular times. Routing is of course a very common problem and many algorithms are available for solving it. But our case is somewhat different from most routing problems. It is actually very dynamic. This means that the timetable is updated at a very high pace. Furthermore the requests for routes will be rather frequent. To provide route information real-time, a high computer capacity is needed. In case of a very extended traffic area the problem might be considered for a distributed approach. This means that the routing algorithm is divided over several parts of the network that do not know the entire state of the network. This way extra computer power can be gained by using several computers.

1.2.5 ABC-algorithm

The Routing system will use an ant algorithm (Ant Based Control or ABC-algorithm). This algorithm uses some of the same principles that living ants in nature do. Although these ants only perceive a very small part of their environment, they can find their way back from the food to their nest. And these animals have a remarkable way of finding the *shortest* way between the food and the nest. They use a pheromone trail, which they can lay and sense. This is explained in further detail in section 2.1.2. Implementations of these principles have shown to be very promising. The algorithm is especially suited for the proposed distributed approach of the Routing system. Conventional algorithms for finding shortest paths, like Dijkstra's algorithm [Cormen 2000], are designed for non-distributed

networks. These algorithms assume that all information about the network is stored at one place. Nevertheless these algorithms may possibly be adjusted for a distributed approach.

1.3 Project description

The different phases done in this project are enumerated below.

1. Study of navigation systems and routing algorithms.
2. Design of a model for a traffic simulation.
3. Design of a model for a routing system
4. Implementation of a traffic simulation program.
5. Testing the traffic simulation program.
6. Implementation of the routing system with ABC -algorithm.
7. Testing the routing system together with the traffic simulation program.
8. Experimenting with the routing system.
9. Evaluation of the experiments.

1.4 Report structure

This report has the following structure. Chapter 2 gives some background information about navigation systems and routing algorithms. In chapter 3 the results of the design phase are presented. Chapter 4 shows technical details of the implementation. Chapter 5 explains the possibilities of the applications that were built. In chapter 6 we describe some experiments. And in chapter 7 we derive the conclusions from the results and provide some recommendations.

Chapter 2: Theoretic background

The background presented in this chapter contributes to the complete understanding of the subject matter that is discussed in the rest of this thesis.

2.1 Routing algorithms

Here we will discuss the two routing algorithms used in the project. Dijkstra's algorithm is a centralised routing algorithm for a static environment with guaranteed shortest paths. A centralised router assumes that all data is available at one location. The ABC algorithm is a decentralised routing algorithm for a dynamic environment with no guarantee for absolute shortest path. A decentralised router can compute route even if the data is distributed over several locations.

2.1.1 Dijkstra's algorithm

Finding the shortest path is a very common problem. E.W. Dijkstra made a great contribution to solving this problem with a path finding algorithm. This algorithm is carefully described in [Cormen 2000]. Dijkstra's algorithm solves the single-source shortest paths problem on a weighted, directed graph G , with vertices V (nodes) and edges E (links). It assumes that all weights (or travel times) are nonnegative and static. The algorithm finds the shortest paths from one source in a graph to all destinations. Of course the shortest paths from all sources to all destinations can be found by repeating the execution of the algorithm for all sources.

Dijkstra's algorithm maintains a set S of vertices whose final shortest-path weights from the source s have already been determined. For those vertices v from S the shortest-path estimate $d[v]$ is set to the shortest path from s to v . The algorithm repeatedly selects one of the remaining vertices not in S with the minimum shortest-path estimate. This vertex is added to S and all edges leaving this vertex are relaxed. Relaxing is testing whether the shortest-path estimate of a vertex can be lowered by designating the current edge as the best edge to reach this vertex. If the test is true, then the shortest-path estimate $d[v]$ is lowered and the best edge to reach this node is set to the current edge. The latter is done by setting the predecessor property of the vertex $p[v]$ to be the vertex on the other end of the edge. In the following pseudo-code implementation of Dijkstra's algorithm we maintain a priority queue Q that contains all vertices V not in S , together with their shortest-path estimate. The implementation assumes that a set of adjacent vertices is available for every vertex.

<pre>1 procedure Dijkstra(G, w, s) 2 begin 3 InitializeSingleSource(G, s) 4 S := CreateEmptySet() 5 Q := CreateEmptyList() 6 AddToList(Q, G.Vertices) 7 while (not IsEmptyList(Q)) do 8 begin 9 u = ExtractMin(Q) 10 AddToSet(S, u) 11 for each v in Adj(u) do 12 begin 13 Relax(u, v, w) 14 end 15 end 16 end 17 end</pre>	<pre>18 procedure InitializeSingleSource(G, S) 19 begin 20 for each v in G.Vertices do 21 begin 22 d[v] := ∞ 23 π[v] := nil 24 end 25 d[s] := 0 26 end 27 28 procedure Relax(u, v, w) 29 begin 30 if (d[v] > (d[u] + w(u, v))) then 31 begin 32 d[v] := d[u] + w(u, v) 33 π[v] := u 34 end 35 end</pre>
--	---

Figure 2: Dijkstra's algorithm in pseudo code

In line 1 we see that Dijkstra is called with three parameters: G is a directed graph with a set of vertices and edges, w is a function that determines the weight between two adjacent vertices, s is the source vertex from the set of vertices. The procedure `InitializeSingleSource` sets all shortest-path estimates to infinity and the predecessor vertices to nil. After that the shortest-path estimate for the source s is set to zero (going from s to s takes no time). In line 4 S is initialised as an empty set. And in line 5 the empty list Q is created. Thereafter Q is filled with the vertices of the graph G . The while loop from line 7 is repeated until all vertices are moved from Q to S . In line 9 the vertex with the lowest shortest-path estimate in Q is assigned to u and removed from Q . This vertex is then added to S in line 10. In lines 11-14 all edges (u, v) leaving u are relaxed. This is, the shortest path estimated $d[v]$ and the predecessor $p[v]$ are updated if the shortest path to v can be improved by going through u .

This algorithm can very well be used for routing problems. It computes absolute shortest paths. For large and dynamic networks it may however be time-consuming, because it has to start all over again when the data is refreshed. This may be a problem in real-time situations. It is also a centralised router: it assumes that all data is available at one place. This is a difficulty for routing in (distributed) networks.

2.1.2 Ant Based Control

The neurones in our brains are only capable of stimulating or inhibiting other neurones when they are stimulated themselves. Still our brains, which are composed of neurons, enable people and other animals to do amazing complex things. The same idea goes for a lot of natural living systems. Complex characteristics emerge from the interactions between relatively simple units and their environment. This behaviour is most often called emergent behaviour. Ants show this behaviour as well. A group of ants does not have a central controlling authority that tells them what to do. They have only local knowledge because of their interaction with the environment. They simply change direction for instance when they find a large obstacle in their way. But they also communicate indirectly with each other through the environment, which is called stigmergy. Many other social insects also use stigmergy. One of the emerging abilities of a group of ants that use stigmergy is to find the shortest route from their nest to a food source [Franks 1989].

Ants only react to local stimuli from the environment but they can also change those local stimuli. Such a modification will influence future actions of other ants at that location. There are two types of modifications. The first type is called sematectonic. Modifications of this type change physical features. An ant digging a hole is an example of this. Ants following the first ant perceive a changed environment, which can cause them to expand the hole. The cumulative effect of such small changes can result in complex structures. The environment can also be changed without contributing to the goal directly. This is called sign-based stigmergy. The goal of sign-based stigmergy is to influence subsequent behaviour. Ants are very good in using this second method of changing the environment. They lay a special sort of volatile hormone, or pheromone, to create a signalling system between ants.

Laying and sensing pheromones

Ants use the laying of pheromone to find the shortest path from their nest to a food source and vice versa. Consider the following situation (Figure 3) where two ants are looking for food. They can reach an apple via two alternative routes. The southern route is shorter than the northern route. The first ants that have to choose a direction have no knowledge about the shortest route, so they choose randomly. One ant takes the northern route; the other uses the southern route (see Figure 4). On their way they lay a pheromone trail, which diffuses slowly. The ant taking the shorter route reaches the food quicker and returns earlier to the nest. This is shown in Figure 5. Here you also see a third ant looking for food. This ant will sense the pheromone and prefer the route with the stronger pheromone trail. The southern route will have a stronger pheromone trail because one of the ants already passed twice, while the other route is only passed once. Consequently the ants will more often use the shorter paths. Some ants will however use other routes sometimes, because there is still pheromone and because they

make errors. This ensures them to check other routes, which may be shorter in the future because of a changed environment.

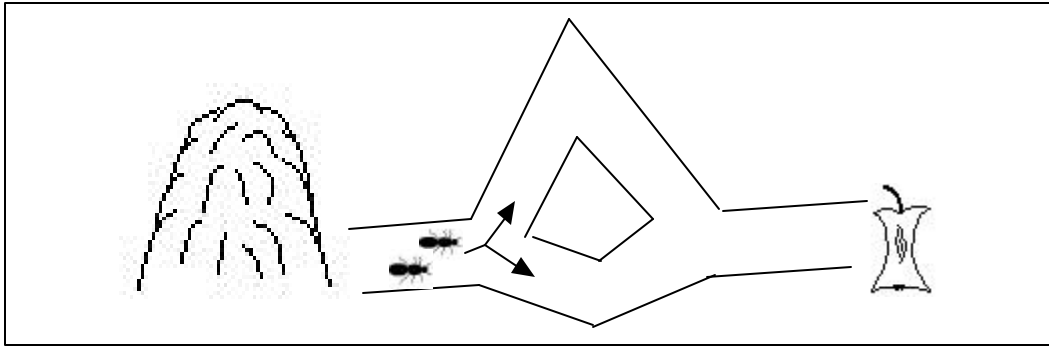


Figure 3: Ants choosing a random route

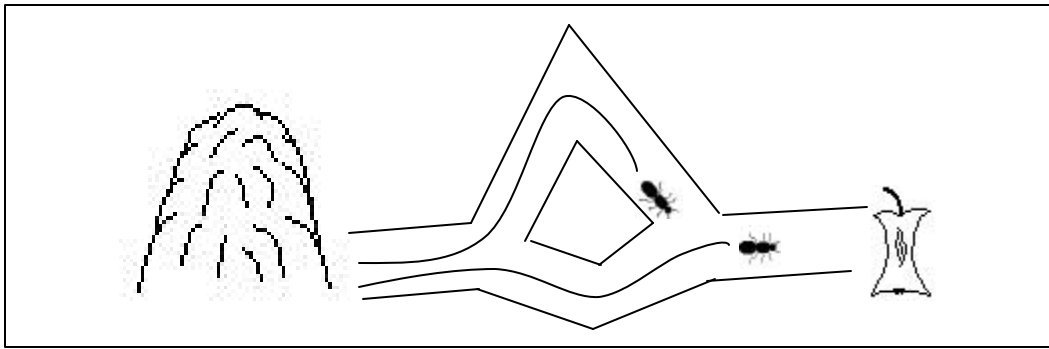


Figure 4: The ants lay a pheromone trail

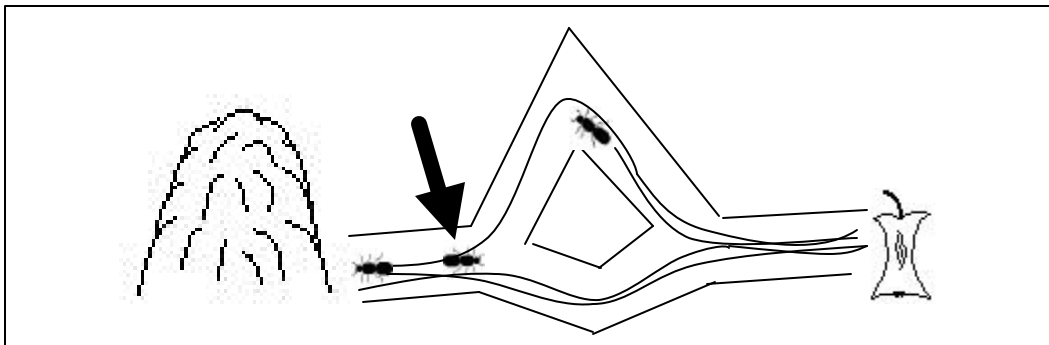


Figure 5: A third ant chooses biased

Reasons for choosing the shortest path

There are mainly three reasons why the pheromone on shorter paths is stronger and the ants choose biased for shorter paths:

- *Earlier pheromone:*
Shorter route will be completed earlier. So the first pheromone trails will attract ants to follow shorter routes.
- *More pheromone:*
The ant density will be higher on shorter routes, causing a merged pheromone trail of more ants. Even if an equal number of ants are taking one of two alternatives, the shorter alternative will have a higher ant density and thus a stronger pheromone trail.

- *Younger pheromone:*
When an ant finishes a shorter route the pheromone is still younger. So it is less diffused and stronger, attracting more new ants.

Ant Based Control algorithm

The idea of emergent behaviour of natural ants can be used to build routing tables in any network. Others have already applied this idea to routing in communication networks. Schoonderwoerd made an application for telephone routers [Schoonderwoerd 1997]. Van der Put applied the algorithm for routing faxes [Van der Put 1999]. Knibbe gives some suggestions for improvement in [Knibbe 1999]. Di Caro and Dorigo have applied it to routers for the Internet [Di Caro 1998]. And Tatomir has compared several versions of the algorithm in [Tatomir 2002]. We will explain the algorithm with the aid of a traffic network in a city, i.e. the composition of the roads and their intersections. Such a network can be represented by a directed graph. Each node in the graph corresponds to an intersection. The links between them are the roads. Mobile agents, whose behaviour is modelled on the trail-laying abilities of natural ants, replace the ants. The agents move across the network between randomly chosen pairs of nodes. As they move, pheromone is deposited as a function of the time of their journey. That time is influenced by the congestion encountered on their journey. Instead of real pheromone a probability is used for all the alternatives. A high probability represents a lot of pheromone. The sum of the probabilities for all alternatives (from one node to one destination) is one. For every possible destination separate probabilities are used. This could be compared to the existence of different kinds of pheromone for different food sources. The agents select their path at each intermediate node according to the probability for each alternative next node. The probability of the agents choosing a certain next node is the same as the probability in the table.

The probability tables only contain local information and no global information on the best routes. Each time an agent visits a node the next step in the route is determined. This process is repeated until the agent reaches its destination. Thus, the entire route from a source node to a destination node is not determined beforehand.

Agents are launched at each node with regular time intervals with a random destination node. They travel around the network using the probabilities in the probability tables. The probabilities per destination are all filled with equal values for all nodes before the process begins. So initially the ants will walk randomly. When an agent finds his destination he will go back to its source via the same way it came from. On its way back it changes the probabilities for the destination it just came from. Short paths get a big update and longer paths receive less update. This way the probabilities for shorter paths will increase faster. When the probability for one alternative is incremented the probabilities for the other alternatives are decremented. This should keep the sum to one. This process produces probability tables, from which the shortest routes to any destination can be read.

Figure 6 is a simple network with 7 nodes. A possible probability table of node 2 is shown in Table 1. There is an entry for every possible destination and every neighbouring node. When traffic needs to be routed the highest probability in the row with the desired destination is taken. The next node corresponding to this probability is used to route the traffic. For example traffic in node 2 with destination 6 will be routed via node 5, because this node has the highest probability for this destination.

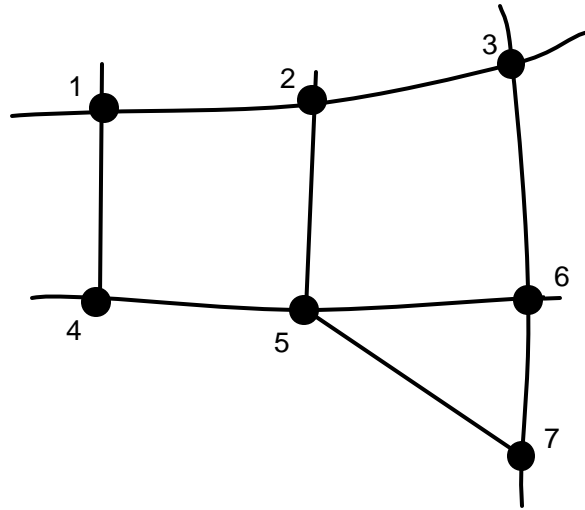


Figure 6: A simple network

Table 1: The probability table of node 2

(2) Destination	Next	1	3	5
1		0.90	0.02	0.08
3		0.03	0.90	0.07
4		0.44	0.19	0.37
5		0.08	0.05	0.87
6		0.06	0.30	0.64
7		0.05	0.25	0.70

2.2 Navigation systems

We will discuss some currently available car navigation systems in this section. Until now only the German Tegarion Scout offers true dynamic route planning, but only on highways. Some other route planners offer some kind of dynamic planning. They let the user select roads that can be blocked or roads may be blocked automatically using traffic information from the radio. This way alternative routes are computed. No good comparison can be made this way, since blocked roads are not taken into account. It may very well be possible that the traveller is advised a longer route than the route via the congested road.

2.2.1 Travelstar

The use of dynamic route planners will result in a better use of the available capacity of the road network, which would result in less congestion. This is something the Dutch government also realises and therefore they stimulate the work of the company Ars T&TT in the development of the Travelstar [Travelstar]. The Travelstar is a hand-held device that can be placed in the car as shown in Figure 7. The hand-held device is a palmtop computer with touch screen, which can also be used outside the car for other common palmtop tasks like a schedule. The Travelstar uses the Radio Data System with Traffic Message Channel (RDS-TMC) for dynamic data about congestions. Furthermore it uses a GPS-receiver to determine the location of the vehicle. This data, together with a map, is presented on the screen. The map moves with the GPS-location and congestions are visualised in red. The system

does however not provide a route, let alone a dynamic route. And the traffic messages are sent with a delay of at least 10 minutes, and the congestions are provided with a length in kilometres instead of delayed travel times.



Figure 7: The Travelstar

2.2.2 Tegaron Scout

The German company Tegaron Telematics GmbH produces the Tegaron Scout [Tegaron]. Currently the Tegaron Scout is the only known navigation system that actually plans using dynamic data. Just like the Travelstar it uses a palmtop computer, but for the dynamic data it uses a GSM telephone. When a traveller starts his trip he enter his destination. A GPS device determines his current location and together these two are sent to the Tegaron station, using the mobile telephone. There the best route is computed using available dynamic data. This route is sent back to the Tegaron Scout. Here it is presented to the user. It shows the user where he is, using the GPS device, and what direction to go. The Tegaron Scout is shown in Figure 8.



Figure 8: Tegaron Scout

Notice that the route is not updated after the start of the trip. Of course the best route can change because of new congestions for example. To update the route the user has to take the initiative for another request. Or the Scout can be set to automatically check for new routes. Therefore the user has to pay a considerable price per request. Presently the dynamic data is only available for highways.

2.2.3 PROMISE

PROMISE stands for Personal Mobile Traveller and Traffic Information Service [PROMISE]. The objective of the PROMISE project is to provide travellers with easy access to a real-time, position-dependent, and multimodal traveller and traffic information service during their whole journey. The activity is supported by the project participants and the European Commission. Some of the partners are Nokia (Finland), Volvo (Sweden), British Telecommunications plc. (Great Britain), Rijkswaterstaat (The Netherlands), IBM Germany, BMW (Germany), Renault, France Telecom Expertel, and Eutelis (France). The PROMISE Project uses NOKIA 9000 Communicator as a portable terminal (see Figure 9). There is also an in-car device for drivers (see Figure 10). The PROMISE services are also intended for other traffic participants, but for a car driver the services are:

- Route Planning using addresses, service locations such as gas station, car parking, cafe and restaurants
- Route guidance with the incorporation of real-time traffic information
- Parking information and reservations

The project has ended in February 1999. A demonstration phase had been carried out in the six named countries. Although stated as the major aim of the project, no dynamic multi-modal route planner has (yet) been developed. The traveller still has to combine traffic reports with planned routes himself.



Figure 9: Nokia 9000 Communicator



Figure 10: In-car terminal

2.2.4 Smart Road project

The Smart Road project is an initiative from Dewilde, professor at Delft University of Technology [Dewilde 2002]. The project is still in a design phase. In the project it is assumed that the road-surface is provided with smart sensors, also called agents, capable of collecting data about weather conditions and the condition of the road-surface and the traffic. The agents can exchange these data, process them and communicate them with passing vehicles. The agents should be self-supporting using solar cells or magnetic spools, activated by passing vehicles. Local information about congestions can be sent to other agents far away via the network of the agents. The results are sent to the driver as warnings about danger on the road and route information. On the basis of historical data, traffic models and actual data the driver can be informed about congestions. Combined with traffic information from the radio the driver can receive adaptive, dynamic route information. The communication between agents and vehicles can be mutual. The vehicle can send information about its trajectory and the travel times to the sensors. Communication with the driver will be done via speech recognition, speech synthesis and a visual display for the maps. The in-car system should also register the actions of the driver. This way the driver will not only be warned for a slippery road-surface, but he gets a second warning when he does not adjust his speed. Or the driver is warned when he does not follow the route instructions.

Chapter 3: Design

3.1 Global design

In this section we will explain the structure of the system from the viewpoint of the vehicle and its driver. A vehicle is driving through a city and its driver wants to be informed how to get from A to B in the shortest time. The driver enters the address where he wants to go and expects a routing system to tell him where to go. Besides the destination the Routing system needs to know the location where the vehicle is at the moment. Therefore the vehicle picks up signals from GPS-satellites (Global Positioning System). This is shown by arrow A in Figure 11. GPS is a system for determining a position with an accuracy of a few meters. This position is measured in latitude/longitude coordinates. In the vehicle these coordinates are translated in a position on a certain road with the aid of a digital map of the city. Now the vehicle has enough information to request the Routing system what route to follow. The vehicle sends its position and its desired destination along with the request for the route to the Routing system (arrow C). Arrow D is the answer from the Routing system that contains the route that the vehicle should follow. These steps are pretty obvious, but we have skipped arrow B. This arrow indicates that the vehicle provides the Routing system with information about the route it has followed since the previous time. The information consists of (1) the location and time at the moment of the previous update, (2) the location and time at this moment and (3) the route that the vehicle has followed in between these times and locations.

All communication between the vehicle and the Routing system can be done with packet switched communication. For this purpose a handheld computer or a dedicated navigation system could be used by the driver. This device should enable the driver to enter a destination and it must be able to present the route to be followed. Furthermore the device should be able to derive a position on a road from the given GPS-coordinates. Table 2 shows a detailed enumeration of the information that is sent along the indicated arrows (A, B, C and D).

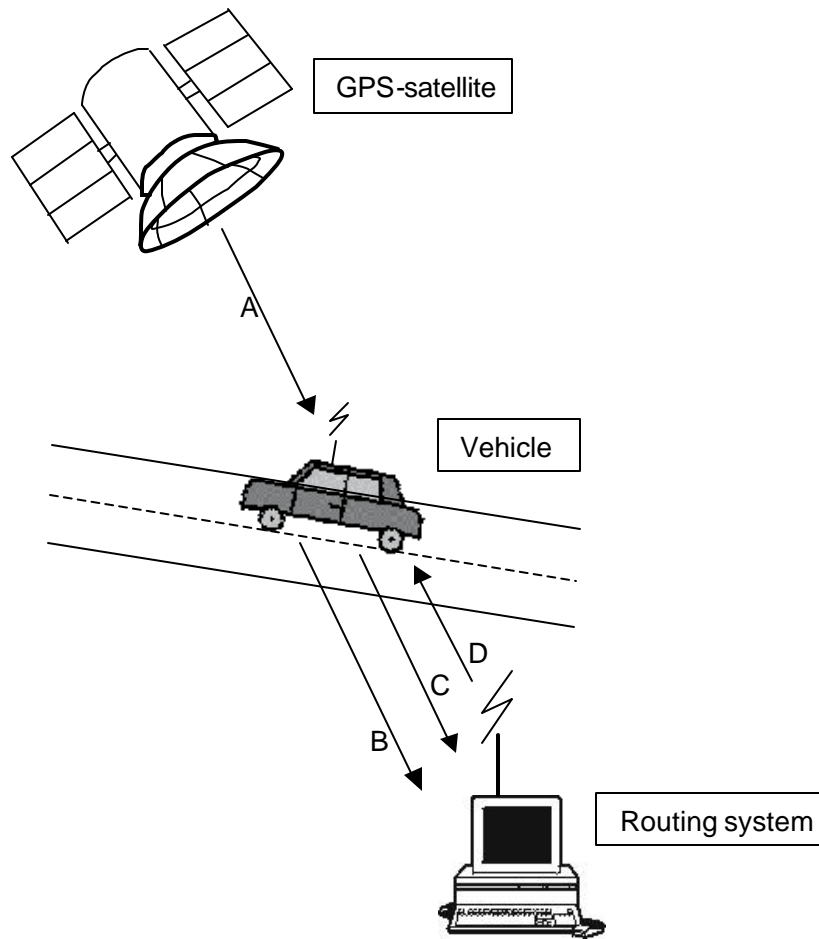


Figure 11: Communication of the vehicle

Table 2: Communicated data between the different objects

	From	To	Data
Arrow A	GPS-satellite	Vehicle	SEND_POSITION, latitude/longitude co-ordinates
Arrow B	Vehicle	Routing system	UPDATE, previous time/position, covered road A, covered road B, covered road C, ..., current time/position
Arrow C	Vehicle	Routing system	REQUEST_ROUTE, current position, destination
Arrow D	Routing system	Vehicle	ANSWER_ROUTE, road A, road B, road C, ...

Communication in time

Now we will give an idea of how the communication works out in time. Firstly, a SEND_POSITION (arrow A) will be picked up from the GPS-satellites. The information that is sent with an UPDATE (arrow B) is most valuable as soon as a new position is known. The fact is that the UPDATE must be sent together with a current position, and the longer it is delayed the older the data is. Therefore an UPDATE will always be sent directly after the current position is known. This does not alter the fact that an UPDATE does not have to be sent after every SEND_POSITION. This frequency can for example be lower to reduce communication. Finally a REQUEST_ROUTE (arrow C) and an

ANSWER_ROUTE (arrow D) will also always succeed a SEND_POSITION. When a new route is requested and acquired, the old route is overwritten. Clearly a REQUEST_ROUTE needs a current position. An old value for the position could cause a vehicle to be travelling on a road that is no longer in the list of its new route. The interval by which a REQUEST_ROUTE is sent can differ from the former messages. After the first route is acquired it will most often be valid for the rest of the travel time. So the frequency can be lower than the frequency of the other messages. Figure 12 clarifies again which messages succeed each other. The time intervals could be adjustable for a driver, but in our prototype they are given a default value.

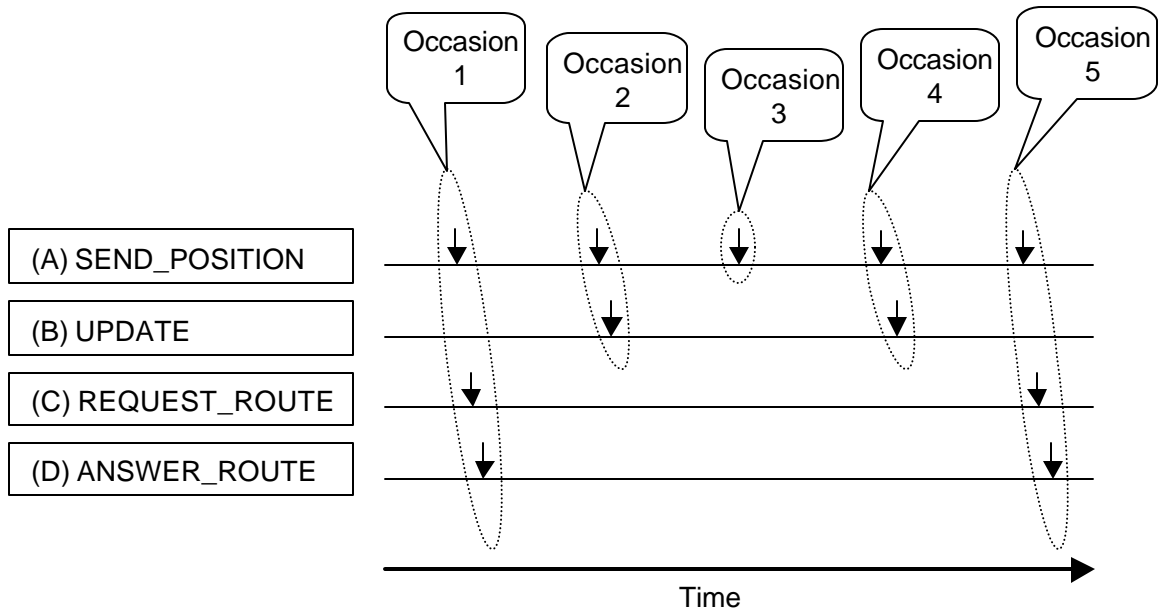


Figure 12: Succeeding messages

The occasions in Figure 12 are illustrated by the trip of a driver in Figure 13. At occasion 1 the vehicle determines its coordinates and requests a route, which is returned by the Routing system. At occasion 2 the vehicle sends information about its trip to the Routing system, after retrieving its location. At occasion 3 the position of the vehicle is determined without any succeeding messages. This should be done to track the route of the vehicle. This way the vehicle knows where it is and when to inform the driver to turn left, for example. Occasion 4 is the same as occasion 2: the vehicle sends new information about its trip to the Routing system. Occasion 5 is the same as occasion 1: the driver receives an updated route to follow.

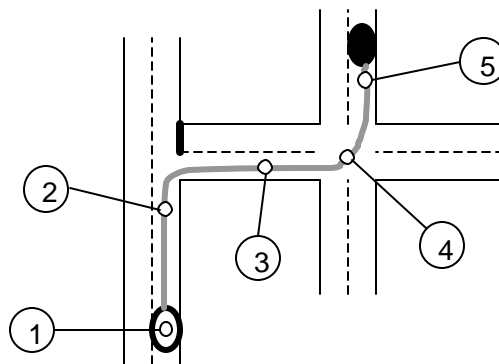


Figure 13: Some occasions during the trip of a driver

3.1.1 Distributed routing system

The use of the ABC-algorithm allows the Routing system to be distributed. This means that the computation is done on several computer systems that are mutually connected via a network. Distribution of computational power gives some advantages above a central routing system. Firstly what normally has to be done by one computer system is now done by several computer systems, which increases the speed and the memory space. Secondly, when properly implemented the failure of one of the systems does not have to imply a total break down of the Routing system. This does however involve some extra communication necessary for information that is not available on the concerning computer system. It will eventually depend on the amount of information that needs to be communicated between the computer systems whether the actual speed will be higher than with a central routing system. Another possible disadvantage is that the traditional routing algorithms cannot be used. But the results of this thesis will have to show that the ABC-algorithm is sufficiently accurate to route the traffic.

3.1.2 Vehicle connection

For a vehicle to acquire the route information it has to communicate with the Routing system. Because of the distributed network we can consider two possibilities to connect with the system: directly to one of the computers in the network or via a server that sends the information to the right computer.

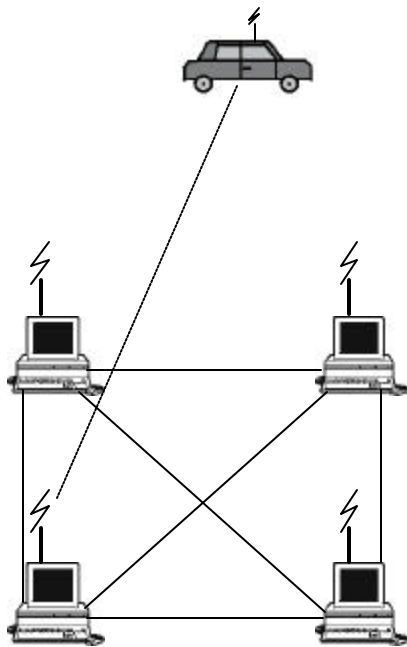


Figure 14: Direct connection

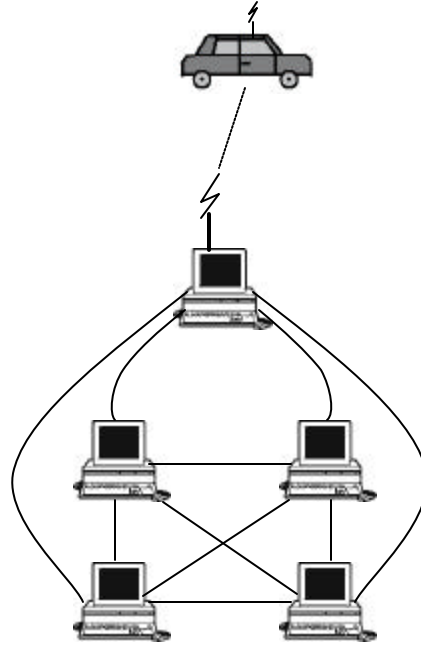


Figure 15: Connection through server

Direct connection

The vehicle could connect directly to the computer system that is the closest in distance or contains (the most of) the information that the vehicle needs to be routed. This is shown in Figure 14. This method is comparable with the way cellular phones work. A cellular phone in GSM-network chooses to connect to the base station with the strongest connection. An advantage of this method is that the communication can always go over a short distance. Furthermore the load of the communication is distributed over different computer systems. And last but not least, the failure of one part of the system

does not have to lead to a breakdown of the entire routing system. On the other side there also have to be more receiving stations and the technology for the connection will be more complex.

Connection through server

The vehicle could also connect to a central server. This server then takes care that the request is sent to the appropriate computer. There the request is handled and the answer is sent back to the server. The server again sends the answer back to the vehicle. This is shown in Figure 15. This method only requires one receiving station and can make use of simpler technology for the connection to the vehicle. A disadvantage is the long range of the connection. And all communication will go via one computer system, which causes an extremely heavy load of that system. And a failure of the central server may cause a complete breakdown.

Used connection

In our case we make use of a simulation where all the vehicles will run on a single computer system. From here the connection goes to the appropriate system that takes care of the routing. So this is most similar to the choice to let the connection go through a server.

3.2 System design

What we want is to optimise the routes for vehicles in a traffic network, so that the sum of the travel times of all individuals is minimal. To test such a system we need information from every driver in the network. Therefore we have designed a simulation of a traffic network with driving vehicles. The challenge is to simulate traffic in the traffic network of a city as well as possible. Once we have the information about the positions and times of individual car drivers we can solve the problem of routing individual drivers. The challenge is to optimise the routes for vehicles in that network given a limited amount of information about the vehicles.

Figure 16 shows a very global picture of the complete system. The main application is the City program. This is where the simulation environment, the City traffic, will run. In this environment vehicles will drive through the road network. The movement of vehicles is influenced by other vehicles and their surroundings. While driving around, the vehicles may send information to the Routing system. There the information is handled by the Timetable updating system. This system processes the information and stores it in the timetable. The Route finding system uses the information in the timetable to construct optimal routes, which can be used by the vehicles of the city traffic. The vehicles send a request for an optimal route, which is answered by the Route finding system. Furthermore there is the Control centre. This is the place where about all the information is showed, from the City traffic as well as from the Routing system. Moreover everything that is adjustable can be adjusted here. These changes can affect the City traffic as well as the Routing system.

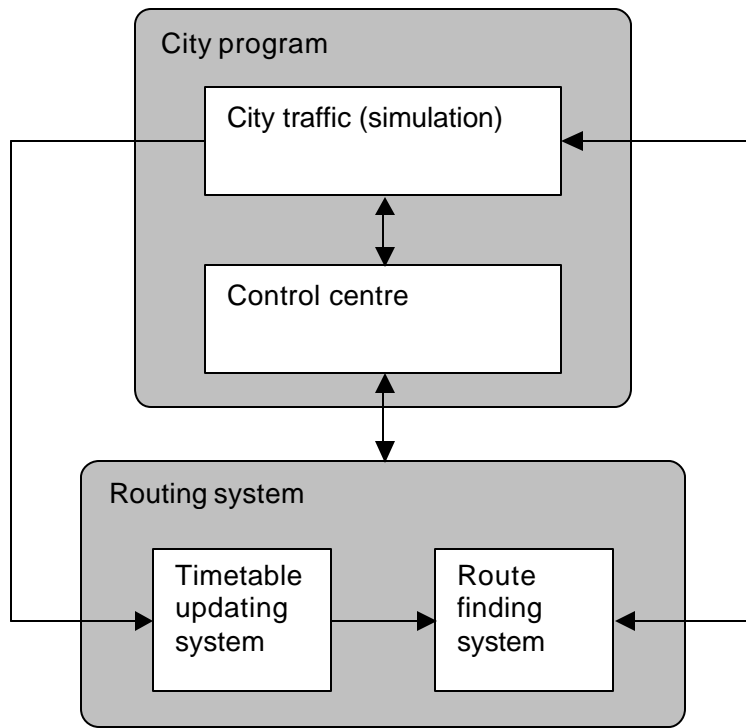


Figure 16: System design

3.3 City program

The simulation is a map of realistic traffic behaviour in a city onto a simulated network. In our simulation a part of the vehicles uses the Routing system to find their way. The other vehicles use fixed routes to get to their destination. A good simulation of the traffic network will have to find a balance between realism and speed. We want the simulation to be realistic so that the Routing system will not have a significant different impact on a real life traffic network. On the other hand the simulation needs to generate data for the routing algorithm at a speed that makes it possible to run several tests. The most important requirements of the City program are:

- *Realism*: it is very important that the impact of the Routing system on the traffic in the simulation will be comparable to the impact on real life traffic in a city.
- *Speed*: the simulation should be fast enough to run several test on the available computer systems.
- *Adaptability*: it must be easy to change the settings of the simulation.
- *Expandability*: it should be easy to improve realism when the quality of the simulation proves to be insufficient.
- *Hardware/software*: it should run on a PC, using Windows NT/2000 as operating system.
- *GUI (Graphical User Interface)*: the traffic network and the vehicles should be displayed for visual inspection.

These requirements will be further explained in the next paragraphs.

Realism

In the simulation vehicles are placed on a city network. The city network consists of links and nodes, representing the roads and crossings respectively. All the vehicles have an individual source and

destination for the route they have to follow. That route should go via links that do not differ too much from the roads that drivers will normally take. Some of the vehicles will be routed by the Routing system. And some vehicles should provide the Routing system with information about a part of the route that they covered and in what time they covered that part. These vehicles that update the Routing system could, but do not have to, be the same vehicles as the ones that are routed. The number of vehicles that are placed on the network should have some realistic distribution over the links. This should ensure that heavily loaded links correspond to heavily loaded roads and the same goes for quiet links and roads. Furthermore it should be possible to change the overall load of the network in time. This enables to simulate rush hours. Different intersections or crossings in the city will have a different impact on the vehicles in the simulation. One can think of normal crossings with priority roads, crossings with traffic lights and roundabouts. Links should be different corresponding to different maximum speeds allowed on the roads and the number of lanes.

Speed

A useful simulation must be able to run in real-time. This is needed for frequency of the messages, which the vehicles send and receive, to correspond to a real life system. But when we run a simulation it must cover a considerable part of the day, for example four hours. We would very much like the simulation to run faster than real-time, so we can run several simulations in the same amount of time. Therefore the clock speed should be adjustable to be able to execute the simulation in real-time as well as accelerated.

Adaptability

It must be easy to change the settings of the simulation. Changes to the network should be simple. The load of the network and the distribution of the vehicles must be adjustable. And the load and distribution can vary over time. The portion of the vehicles that send update information, the portion of the vehicles that request route information and the portion of the vehicles that do both will have to be adjustable.

Expandability

It should be easy to improve realism when the quality of the simulation proves to be insufficient. One can think of the speed of the vehicles or the handling of the vehicles at intersections. These are two important topics that could need extra attention to improve the resemblance between the simulation and the real life situation. One should keep in mind that such improvements often go together with a decrease of speed.

Hardware/software

We expect the City program to run on a common PC with Windows NT/2000. This is one of the most commonly used systems nowadays. This will allow others to experiment with the software, without the need of specialised hardware or software.

GUI

The City program requires a visual representation of the traffic network and the movement of the vehicles. This is one of the best ways to test the quality of the simulation and understand the effects of the Routing system. Furthermore some statistical data is needed represented as graphs. It would be useful to be able to create these graphs from the user interface of the City program.

3.3.1 Vehicle movement

An important component of the simulation is the movement of the vehicles. The vehicles reside on the links. A link is a road from one intersection to the next. All lanes between two intersections that are directed in the same way belong to the same link. The lanes in the opposite direction belong to another link. Figure 17 shows six links. Four links consist of two lanes and two links consist of one lane. The links are divided into a number of blocks. There can only be one vehicle per block per lane at the same time. This ensures a maximum density on the links. The indicated link has two lanes and three blocks. So there can be six vehicles on that link at the same time. The vehicles are moved stepwise. Every step the vehicles will be moved for example 0.2 seconds. So if a vehicle has a speed of 10 m/s (= 36 km/h) then every time step it will be moved 2 meters. The speed of a vehicle depends on the maximum speed assigned to the current road. All vehicles on one road will have the same speed, unless they have to stop for other vehicles or traffic lights.

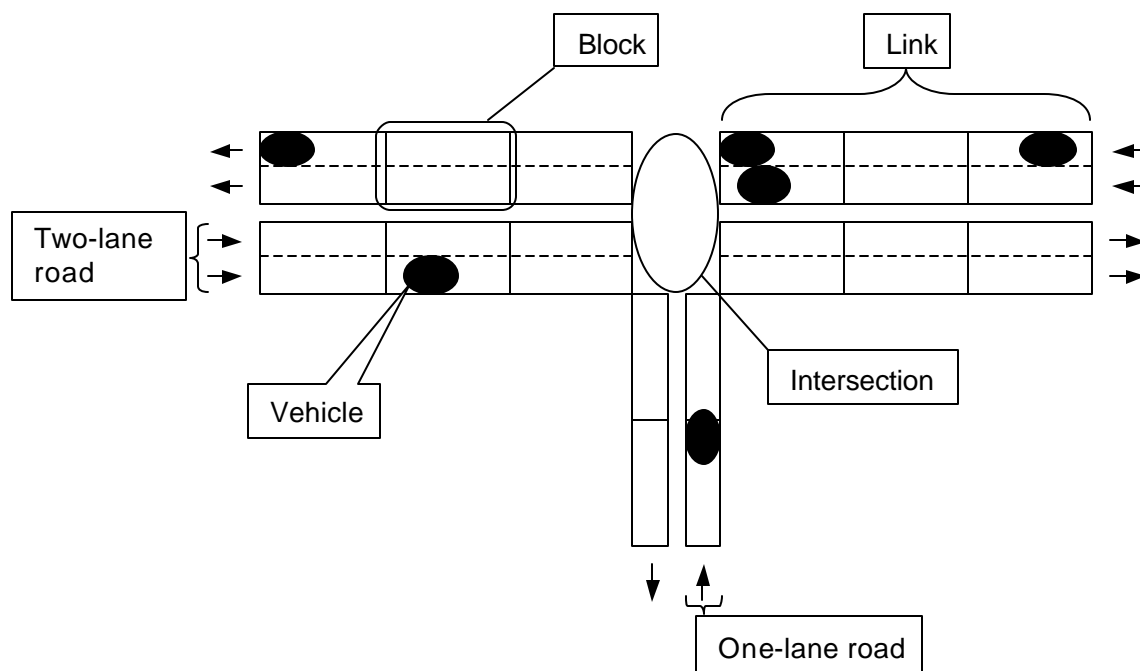


Figure 17: Representation of the traffic network in the simulation

Figure 18 shows the movement of a vehicle in one time step. In this case the vehicle marked with A is moved from position 0 to position 7. Normally the steps are smaller unless the vehicle has a very high speed. But this big step is made to illustrate how to deal with different aspects of the movement. The vehicle cannot be moved from position 0 to 7 without making smaller steps. A number of things have to be taken into account on its way, like obstructing vehicles and intersections for example. There can always be only one vehicle per block (per lane), so the vehicle can be moved to the end of its block without interference. But before going to the next block it should check for an existing vehicle on that block. And prior to being placed on another link, not only the occupation of the block should be checked, but also the colour of the traffic lights and the precedence rules. For this purpose a flow diagram is created (Figure 19).

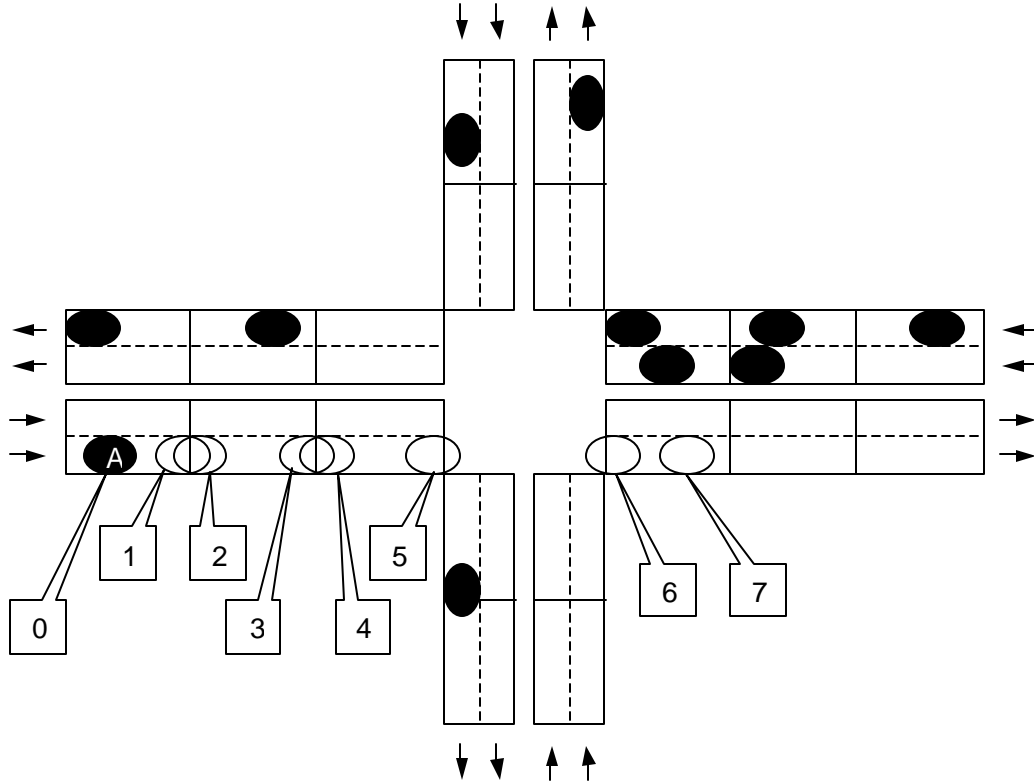


Figure 18: The movement of a vehicle in one time step

The flow diagram in Figure 19 will be explained with Figure 18. Because it is possible that the procedure to move a certain vehicle is called twice per time step, it is checked whether the vehicle is moved this time step before. Directly after that a mark is set to remember that the vehicle is moved. Look again to the vehicle marked with A in Figure 18. The vehicle starts at position 0. According to the size of a time step and the allowed speed on the link, the vehicle can move a certain distance per time step. This is stored in *movedist*. Assuming that the vehicle can move to the end of the block (*blockendpos*) from its current *position*, given the *movedist*, the vehicle moves to the end of its block (position 1). If the next block on the same link and there is no vehicle in the next block it moves further to the first position of the next block (position 2). This is repeated until the vehicle has done its maximum movement for this time step (*movedist* = 0). A special case appears when the vehicle is at the end of a link in front of an intersection (position 5). Then the vehicle does not only have to take into account that there is space on the first block of the next link. It also must be possible to cross the intersection. How this is handled is explained in section 3.3.2. In case of a road with more than one lane, the vehicle will first try to enter the first lane. If it is occupied by another vehicle, it will try the second lane, and so on. Our model does not deal with pre-sorting. When a road consists of more than one lane, no lanes are especially designated for a certain direction, like going left or right. So it does not matter where the vehicle will go, it can be placed on any of the lanes when approaching an intersection. Even if the vehicle is on the first lane and there are more lanes with vehicles, it can turn left without having to wait for the vehicles on the other lanes of the same road.

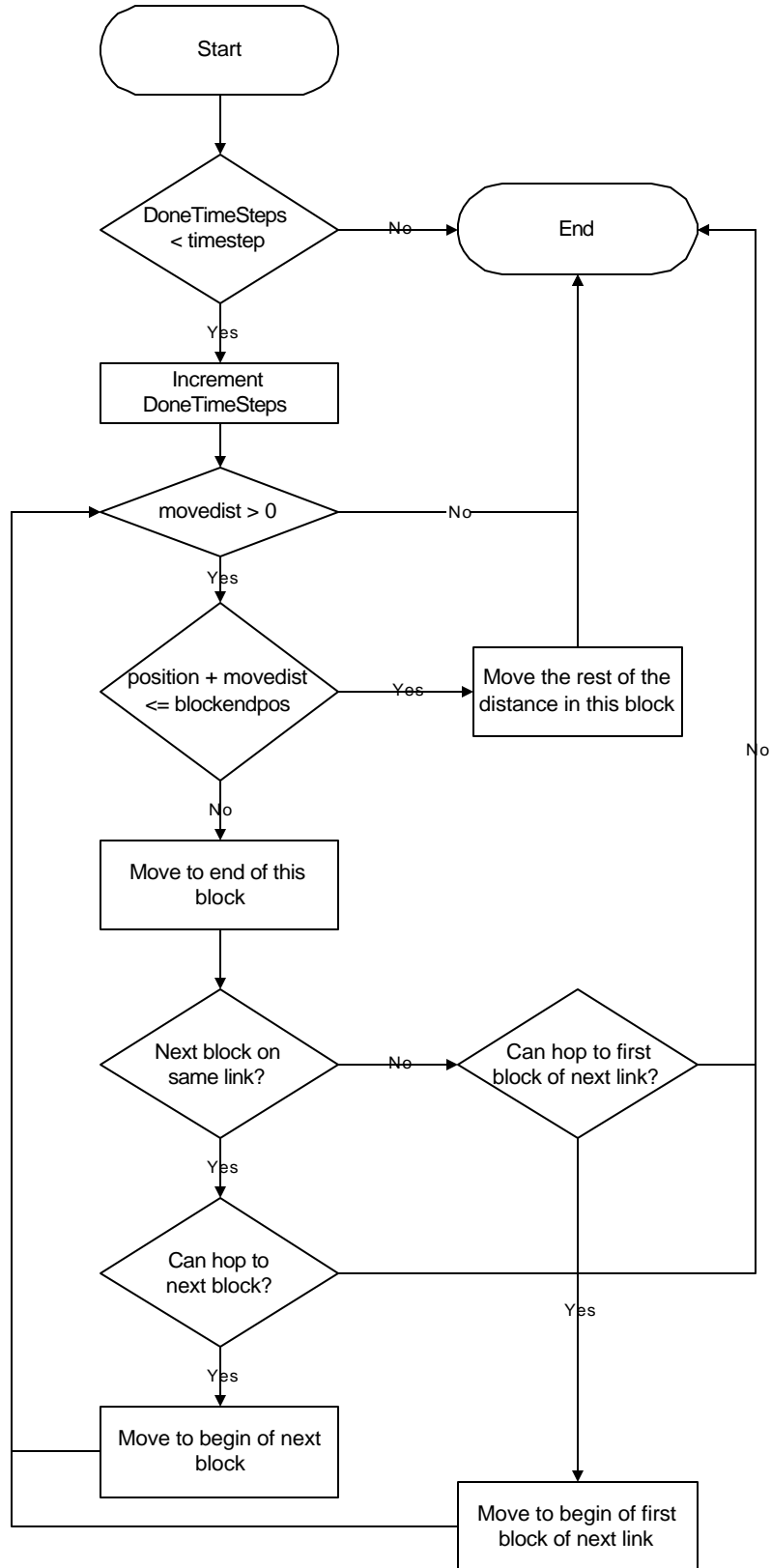


Figure 19: Flow diagram of the movement of a vehicle

Handling order

The fact that the vehicles are moved one by one each time step causes some complications. Imagine that vehicle A follows vehicle B with a distance of one block at a one-lane road. The speed of both vehicles might allow them to move two blocks per time step. When vehicle A is moved first, it cannot move to the next block because vehicle B occupies it. But at the end of the time step vehicle B is moved too and there would have been enough space for vehicle A. This causes an unnecessary gap between the vehicles that would not have been there if vehicle B was moved first. To avoid this problem we could use a complex algorithm to sort the vehicles. But instead of that we use an approach that is less time-consuming. Every time a vehicle tries to hop to another block we just check whether it is occupied by one or more vehicles. And when it is, we just move those vehicles first, before we continue. This way it could possible that a vehicle is moved twice per time-step. To avoid that, we check that every vehicle is moved only once per time-step.

3.3.2 Intersection handling

In our simulation we distinguish three different crossings or intersections. The first one is a normal intersection, where roads with and without priority join. The second one is a crossing controlled by traffic lights. And the last intersection type is a roundabout.

Normal intersection

Most intersections are locations where three or four roads join. Those intersections can further be subdivided in intersections with and without main roads. And the intersections with main roads can be subdivided in crossings with the main roads moving straight on or with a curve to the left or right. The stated intersections are shown in Figure 20. The roads without priority are distinguished by the bold stop line.

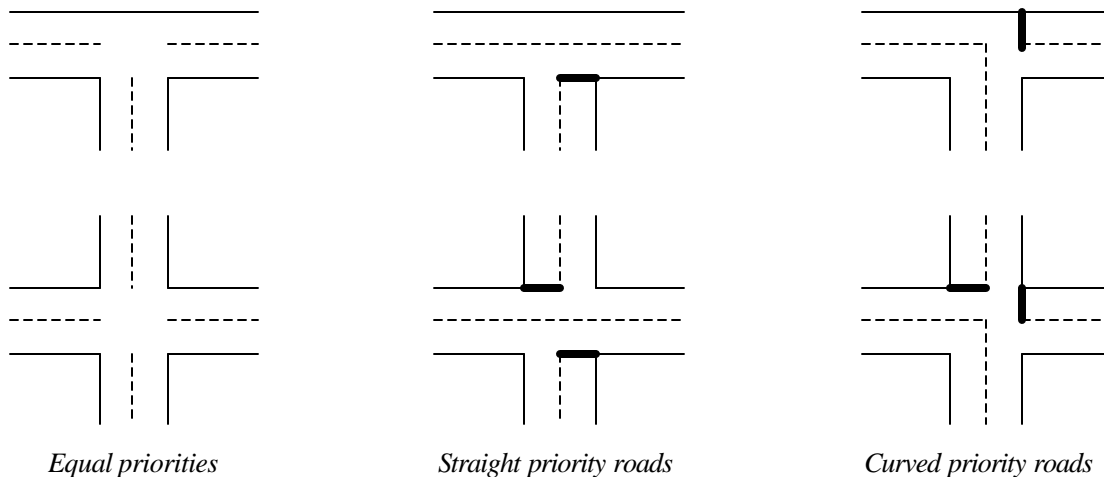


Figure 20: Examples of normal crossings

Although a classification of intersections in one of the shown categories would allow most intersection types to be realised in the simulation, we have chosen a more general approach. This approach allows the usage of intersections with an unlimited number of joining roads and any possible angle for these roads. What is important for the intersections is how the traffic reacts upon it. This means, should a driver wait for another vehicle on a different road and give precedence or take precedence and drive. This problem deals with three roads at a time: the current road of the driver, the road where he is heading for, and one of the roads where other vehicles drives on. For every combination of roads that is possible at a certain intersection, we determine and store the answer to the question whether to give

precedence or not. Of course the precedence rules for an intersection also depend on the vehicles present at the intersection, but that information is dynamic. And therefore those aspects of the application of the precedence rules are postponed to the run-time of the simulation. However the question whether to wait for other traffic can be computed before the start of the simulation. For that purpose we need to know three things:

1. Does the current vehicle drive on a road with priority?
2. Does the other vehicle drive on a road with priority?
3. Is the other vehicle coming from the right?

The first two questions can simply be answered by determining the properties of the corresponding roads. And question 3 can be derived from the geometric properties of the roads. When these questions are answered we can determine whether a vehicles has to wait and give precedence. The general results are shown in Table 3.

Table 3: Intersection state determination

Vehicle from road with priority	Approaches road with priority	Other road is on the right	Wait for traffic
Yes	Yes	No	No
Yes	Yes	Yes	Yes
Yes	No	No	No
Yes	No	Yes	No
No	Yes	No	Yes
No	Yes	Yes	Yes
No	No	No	No
No	No	Yes	Yes

Imagine for example that the roads that join the intersection all have the same (low) priority. A vehicle might approach the intersection from the south and want to go north. When there is a vehicle coming from the east (right), the first vehicle will stop. This corresponds to the precedence rule that a vehicle has to give way to the right. When judging whether there is traffic coming from a certain direction, we do not only look at the vehicles that have already arrived at the intersection, but also at the vehicles that are still some distance away from the intersection. When a vehicle on the main road wants to turn left it will have to wait as long as there is traffic on the road ahead. If a vehicle drives on a road without priority and wants to turn left, it has to wait for all the traffic from ahead and the right. However it only has to wait for traffic from the left if there is a main road. The only problem that might arise is that a deadlock appears. That problem will occur when for instance two vehicles approach from opposite directions and they both want to turn left. Therefore it is checked whether all vehicles in front of an intersection are waiting for precedence, and there is no traffic that is still approaching the intersection or just crossed the intersection. If that is the case the first vehicle in the next time step will get precedence anyway, regardless of the precedence rules.

Traffic lights

The traffic lights in the simulation are controlled with a fixed time control. This means that the green time of a traffic light will be the same every cycle of the lights. Currently the green time is not adjusted to the amount of vehicles waiting at that moment. This functionality can be extended in the future. For now the state of the traffic lights is determined at the basis of a table. This table consists of a time-interval, in which the light should be green, for every light number mentioned in Figure 21. Table 4 is an example of such a table with time-intervals. In this example the traffic from each direction gets 25 seconds of green light per two minutes. There is a vacation period of 5 seconds between two green lights. The lights are red when they are not green. There is no intermediate colour. A vehicle just has to stop when the light is not green. Every time step the state of the intersection is determined with the aid of a traffic table. However, when two opposite directions have a green light for going left at the same time, the cars would possibly cross each other's lane. Therefore the precedence rules used with normal intersections are applied to the green lights only. This way it can

happen that a light is green according to the traffic light table, but the vehicle has to wait for another vehicle with a green light and priority.

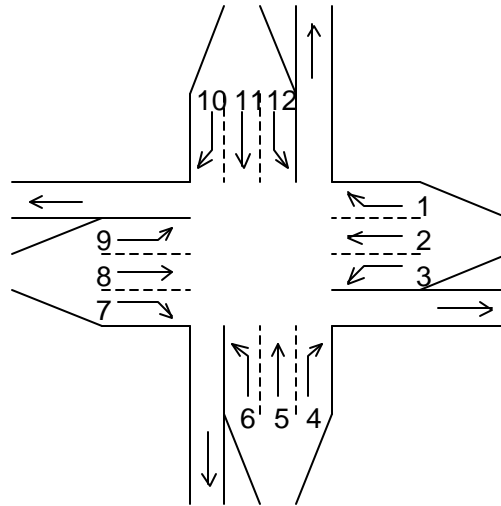


Figure 21: Traffic light numbering

Table 4: Traffic light table

Cycle time: 120 s		
Light number	Start green-time	End green-time
1	0	25
2	0	25
3	0	25
4	30	55
5	30	55
6	30	55
7	60	85
8	60	85
9	60	85
10	90	115
11	90	115
12	90	115

Roundabout

Roundabouts are a very special case. With a normal intersection the decision to cross it depends mostly on the traffic that approaches the intersection. But the flow on a roundabout depends largely on the traffic that is on the roundabout, because vehicles on a roundabout have priority over other vehicles. In any case we will not wait for a vehicle from the right until it has covered a great part of the roundabout and is actually coming from the left. This illustrates that we need to model traffic on the roundabout itself. The easiest way to do this is to model a roundabout like a set of normal T-junctions. See Figure 22.

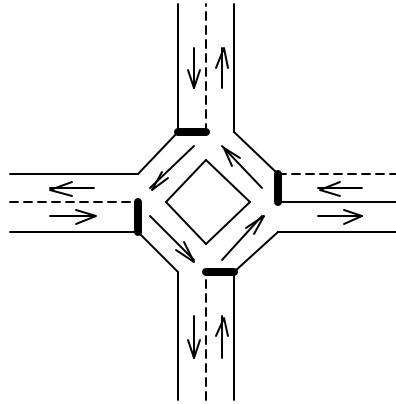


Figure 22: The representation of this roundabout uses four T-junctions.

Needless to say the road parts on the roundabout are one-way roads. The traffic should always turn right when entering the crossing. As the traffic on the roundabout mostly has precedence over the approaching traffic, the road parts of the roundabout are modelled as main roads. The default speed on the roundabout is modelled considerably lower than the speed on the other roads. For the ease of entering the data for a complete traffic network, a roundabout can be entered as one intersection. The input will be converted to represent the required T-junctions. There is however an important drawback of modelling a roundabout this way. An approaching vehicle will wait when another vehicle is driving from the left on the roundabout. When the driver on the roundabout makes clear to turn right and leave the roundabout, the first vehicle may drive. But in our model the vehicle will not drive, because it is unable to detect that the other vehicle will turn. We did not model the use of direction indicators on vehicles. This also applies to the other intersection types, but there it is less apparent. In real life there are still a lot of drivers that do not show correctly when they are turning, for that matter. This causes the same effects of unnecessary delay.

3.3.3 Control centre

The Control centre is a part of the City program that controls both the City traffic (simulation) and the Routing system. Here a traffic simulation can be started synchronously with the Routing system. The speed of both systems may be different however (measured in steps per second). Controlling the simulation and the Routing system also means adjusting parameters used in the procedures. It might be more customary to have a separate control centre as a part of the Routing system. But because of the provided option to have a group of collaborating Routing system parts working as one Routing system, it is more convenient to combine the control in the City program. Otherwise changes in the parameters of the Routing system would have to be entered several times, once for every part of the Routing system. Now we can change both the parameters of the traffic simulation and the parameters for the Routing system in one view. The changes in the parameters of the traffic simulation are sent to the City traffic (simulation). The changes in the parameters of the Routing system are sent to all active Routing system parts.

3.4 Routing system

In this section we discuss the Timetable updating system and the Route finding system. These two subsystems together form the Routing system. The relations are shown in Figure 23. The function of the Route finding system will be clear: we are building a system to route vehicles. The reason why we need the Timetable updating system is the following. The Route finding system needs information about the state of the network. A static system could use a fixed set of data, but we will use a dynamic

system that needs dynamic data. Those data are provided by the Timetable updating system. That information can be for example the load of the parts of the network but a more direct and therefore more practical type of information is the time it takes to cover a road. Vehicles send information about their covered route to the Timetable updating system. From that information this system computes the travel times for all roads and stores it in the timetable in the memory. Besides the timetable also a history of measurements is stored in the memory. This history of measurements has a limited size and old measurements are deleted when they become obsolete. The reason to keep a history will become clear later in this chapter. The Route finding system uses the information in the timetable to compute the shortest routes for the vehicles. When a vehicle requests route information the Route finding system sends this information back to the vehicle.

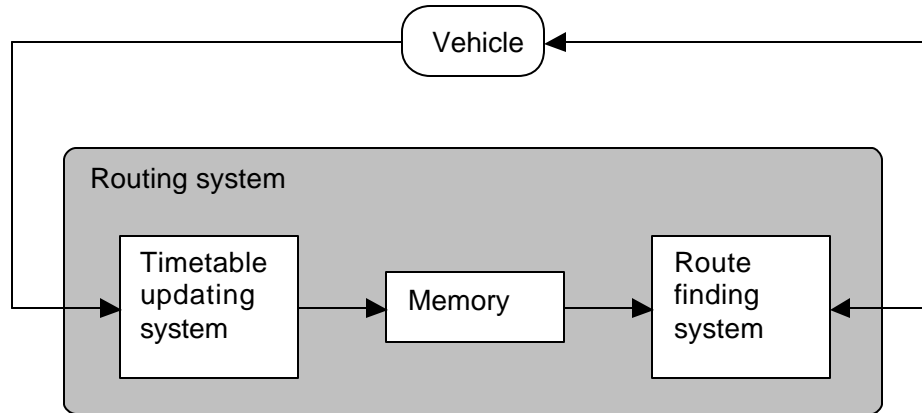


Figure 23: Design of the Routing system

The most important requirements of the Routing system are:

- *Quality*: the Routing system should provide routes that are close to the shortest possible routes in time.
- *Speed*: the Routing system should be able to compute new routes in real-time and even faster for experimental purposes.
- *Expandability*: it should be easy to reflect lasting changes of the city network, like new roads, in the Routing system.
- *Adaptability*: it should be simple to disable roads to generate diversions in cases of roadworks.

These requirements will be further explained in the next paragraphs.

Quality

We expect the Routing system to provide the drivers with routes, which approximate the shortest possible routes in time. It may be clear that guaranteeing absolute shortest routes is impossible for several reasons. First of all there is no complete overview available of the current state of the traffic. Let alone that we can perfectly predict what the traffic participants will do in the near future. We cannot accurately predict where vehicles will go and certainly not whether pedestrians will cross a road. We can only use historic data from the recent past to compute shortest routes.

Speed

The Routing system has to compute new short routes continuously, and on the other hand it has to handle the individual requests from drivers to provide personal routes. The Routing system should be quick enough to handle both these matters in real-time. For the simulation environment however it has to be even faster. Given the point that we want to be able to accelerate a simulation for experiments with multiple runs, also the Routing system must be capable of acceleration beyond real-time speed.

Expandability

The network of roads in a city is subject to continuous changes, like new roads, extra lanes, changing intersections, etcetera. The Routing system should provide simple methods to reflect those changes. It should also be easy to expand the network to a larger environment, for example to combine the Routing system for neighbouring villages.

Adaptability

This term relates to temporary changes in a city network as opposed to the expandability requirement. The Routing system should provide means to disable roads and change maximum speeds, to deal with diversions in case of roadworks, for example. Such short-term changes should be realisable very quickly, to have the Routing system keep providing correct routes.

3.4.1 Timetable updating system

The Timetable updating system could receive its information about the traffic network in the city from different sources. This could for example be directly from sensors in the road-surface. But the main sources are the vehicles themselves. They provide the system with information about the last part of the path they followed and the time it took them to cover it. With this information the Timetable updating system computes the travel times for every link. The travel times are placed in the timetable. This timetable can be seen as a two-dimensional matrix with all intersections of the traffic network along both axes. When one can go from one intersection directly to another, there will be an entry in the table that represents an estimate for the time to cover that road. The intersections that cannot reach each other unless via other intersections will have no entry in the table. Figure 24 is an example of a traffic network and Table 5 shows its timetable.

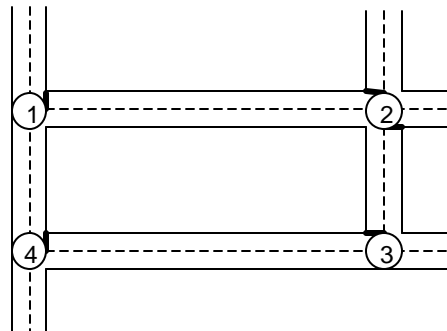


Figure 24: A simple traffic network

Table 5: The timetable from the traffic network of Figure 24

From:	To:	Intersection 1	Intersection 2	Intersection 3	Intersection 4
Intersection 1			25 s		13 s
Intersection 2		26 s		18 s	
Intersection 3			18 s		27 s
Intersection 4		12 s		25 s	

We will now explain how we represent a traffic network in our model. Figure 25 shows a simplified part of a city map. Figure 26 shows the internal representation of that map. As one can see there are a forward and a backward link for every road. This represents that the traffic can move in both directions. Furthermore there can be noticed that at every point, where a driver can choose between several roads, the road is divided into separate links.

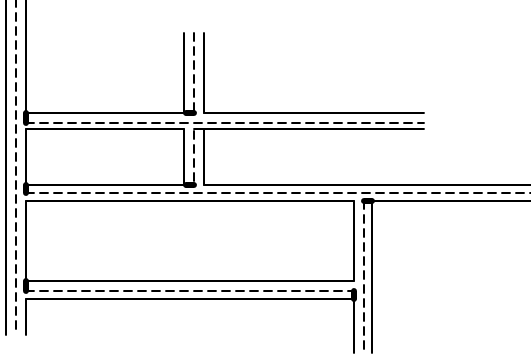


Figure 25: A part of a city network

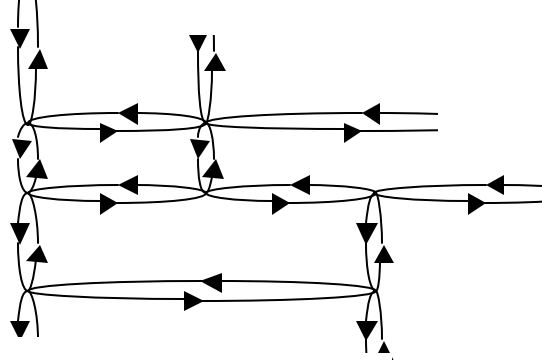


Figure 26: Internal representation of a part of the city network

Now imagine a car driving from position 1 to position 2 over the grey line as shown in Figure 27. At the time of 14:33:15 the car starts driving from position 1. This information is received from the GPS-satellite and temporary stored at the vehicle. 34 seconds later the vehicle again receives its position from the GPS-satellite. This is shown in Figure 27. Now the vehicle sends the first and second time and position to the Routing system (in particular to the Timetable updating system) along with the links it has covered between them. With the help of the available information about the network that is present at the Routing system, the Timetable updating system can compute the time for every part of the covered road. Therefore we need the total of the covered road since the last update of the vehicle to compute an estimate for the time to cover the separate links:

$$\left. \begin{aligned} D &= \sum_l d_l \\ M_l &= \frac{L_l}{D} (t_2 - t_1) \end{aligned} \right\} \quad (1)$$

- d_l is the covered distance on link l ;
- L_l is the length of link l ;
- D is the total road covered by this vehicle since the last update;
- t_1 and t_2 are the times of the updates;
- M_l is the measurement of the time for link l .

In our example the total distance D is 280 meter. $(t_2 - t_1)$ is 34 seconds. So if we look at the link of 50 meter then

$$M_l = \frac{50}{280} \cdot 34 \approx 6$$

Hence we use those 6 seconds as an estimate for the time to cover that link. That time is used to update the timetable. This updating is not straightforward. Aspects to keep in mind are for example:

- Several cars provide new information, but some data is older than other;
- There are no vehicles at all to provide information about certain roads.

How to deal with these problems is explained later in this chapter.

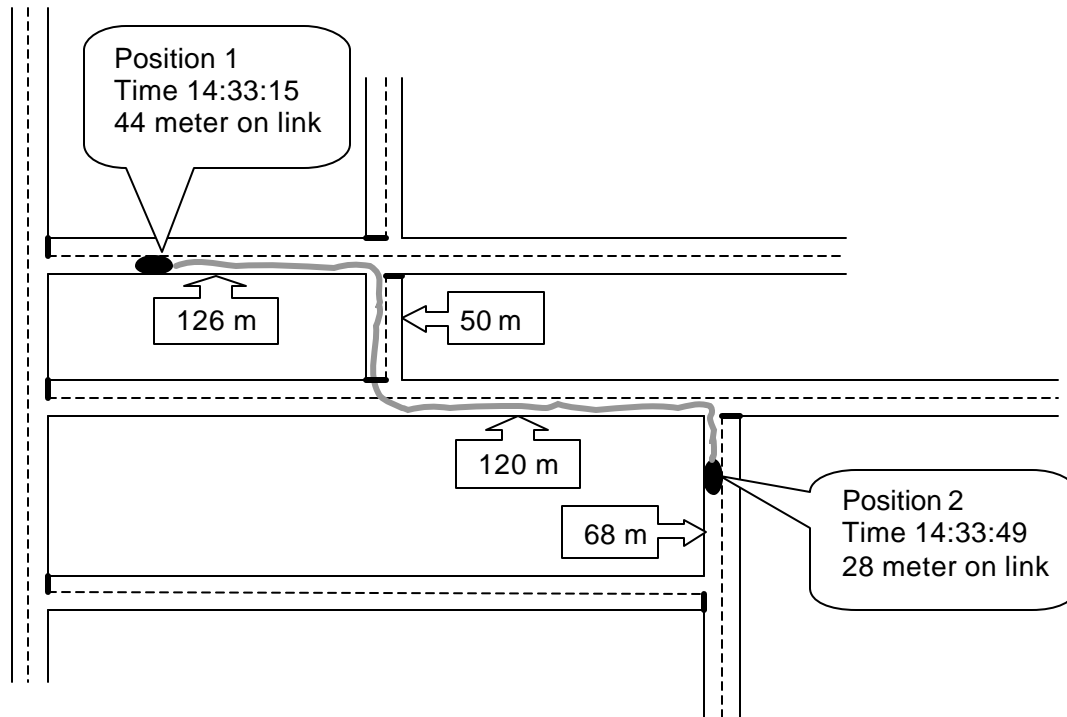


Figure 27: A vehicle driving through a city

Constraints

The most accurate results for a time estimate for a certain link are acquired when the vehicle sends information at every corner of the road. A possible drawback is the amount of data and communication required. A more flexible method is to send the information at certain intervals. But when the vehicle has covered many links before it sends new information, the average is computed over all those links, assuming a continuous speed of the vehicle. And the real value for a single link can deviate a lot from the average value. Furthermore the delay between driving on a link and updating the time estimate of that link might get too long. Another issue is the way the intersections are handled. Turning left or right will slow down a vehicle considerably. But in our model turning is handled the same way as going straight on. Another method is to treat an intersection as several links: one for each source/destination pair. Such an extension could be fit in our model without any adaptations. Another limitation of our model is that it does not take into account that some drivers drive faster or slower on an empty road or a busy road. Besides that, there should be taken care of the fact that drivers do not usually start and end their trip on a road. They leave it to park somewhere, which makes their data unreliable. Or they stop temporary to unload something or someone. In our simulation these problems just do not occur.

Update impulse

On quiet roads it is very well conceivable that there will be no vehicles that send route information for a long time. When there are no updates for a certain part of a road we still want to know an estimate for the time it takes to travel that way. For that purpose we can use the length of the road, the maximum allowed speed and some correction factor for example. The length and speed yield a time

estimate for the road. That time can be adjusted a little with the correction factor to represent that the average speed will be some higher (or lower) than the maximum allowed speed. This yields a default value for the travel time of the road. When we look at a road with a length of 200 meter where the average speed is 50 km/h (= 14 m/s), the default value will be 14 seconds. When at some time a vehicle covers that road in 20 seconds we want the entry in the timetable to be adjusted so that it represents a value of about 20 seconds. This way the routing algorithm will less likely use the road for other vehicles. But half a day later it is useless to know that there has ever been a car that was delayed on that road. The situation has changed many times since then. In fact information older than an hour is usually obsolete. So what we really want is that new information gives an impulse to adjust the time, but the effect should gradually diminish. And after for example an hour the effect of the information should be faded out completely. If closely after the first a second car provides the system with new information, then we should compute a weighed average. The information of the first car counts less than the information of the second, because it is older. And when the information of both vehicles gets a little older the default value should become more important. There are several ways to accomplish this, but we will use the following function because it is intuitively useful and easy to adjust.

$$T(t) = \frac{D + \sum_k W(t - S_k) \cdot M_k}{1 + \sum_k W(t - S_k)}$$

with

$$W(t) = w \cdot c^{t^2} \quad \forall 0 \leq t < h$$

$$W(t) = 0 \quad \text{else}$$

}

(2)

- $T(t)$ is the value in the timetable at time t ;
- D is the default value in the timetable;
- M_k is the result of the k^{th} measurement acquired from the information of a vehicle;
- S_k is the start time at which the information is added;
- $W(t)$ is the weight of a measurement at time t ;
- w is the weight of the measurement at the start time;
- c is a constant that ensures a diminished weight in time;
- h is the period that a measurement has any effect on the outcome.

To explain this function:

As long as there are no results of a measurement, only the default value D influences the value in the timetable. When, at time S_1 , the first measurement is received, that value M_1 gets a weight of w (for example 25), because t in $W(t)$ is 0 when t in $T(t)$ is S_1 . Then the value for the timetable is a weighed average:

$$\frac{1 \cdot D + 25 \cdot M_1}{26}$$

The weight of M_1 fades in time when c is positive and less than one, for example 0.99. With the used function $W(t)$ the result of a measurement on the value of the timetable will be shaped like a half bell. This is shown in Figure 28.

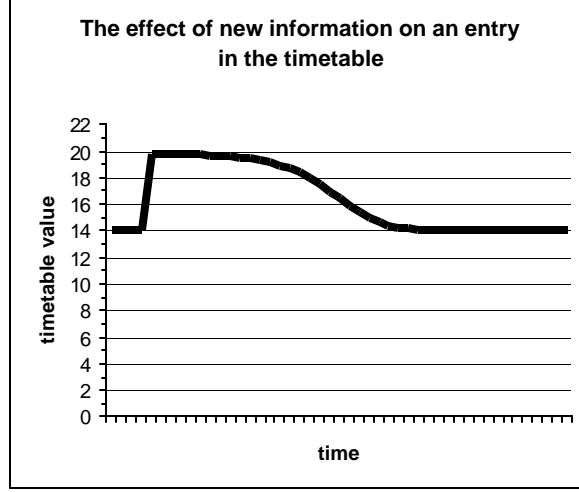


Figure 28: An update gives an impulse to a value in the timetable

After h time a measurement does not have any effect anymore. So all measurements must be kept in memory for h time. The function takes care to average measurements when there are more measurements available for a certain road. When a new measurement becomes available the values should be computed again. But the values also have to be computed at regular intervals because of the fading effect.

3.4.2 Route finding system

An important aim of this thesis is the Route finding system. This system receives requests for routes from individual motorists. For each motorist the shortest route in time will be calculated and send back to the motorist. The Route finding system uses the Ant Based Control algorithm (ABC -algorithm) described in [Van der Put 1999]. This algorithm makes use of forward and backward agents. The forward agents collect the data and the backward agents update the corresponding probability tables in the associated direction. The algorithm consists of the following steps:

- At regular time intervals from every network node s , a forward agent is launched with a random destination d : F_{sd} . This agent has a memory that is updated with new information at every node k that it visits. The identifier k of the visited node and the time it took the agent to get from the previous node to this node (according to the timetable) is added to the memory. This results in a list of (k, t_k) -pairs in the memory of the agent. Note that the agent can move faster than the time in the timetable.
- Each travelling agent selects the link to the next node using the probabilities in the probability table. A random number between 0 and 1 is generated to choose one of the nodes according to the magnitude of the probabilities. The probability for the node, where this agent just came from, is filtered out for this agent. This way the agent will not directly go back to that node. Also the nodes that are reached by disabled links are filtered out. Note that the probability table is not updated yet.
- If an agent has no other option than going back to the previously visited node or it reaches another already visited node, a cycle in its path arises. This cycle is deleted from the memory of the agent.
- When the destination node d is reached, the agent F_{sd} transforms to a backward agent B_{ds} . The backward agent uses the memory of the forward agent to find the way back.
- The backward agent travels from destination node d to the source node s along the same path as the forward agent, but in the opposite direction. It uses its memory instead of the probability tables to find its way.
- When the backward agent steps from some node f back to some node k , it updates the probability table in the current node k . The probability p_{df} associated with node f and destination node d is

incremented. The other probabilities, associated with the same destination node d but another neighbouring node, are decremented. The used formulas are given below.

- When the backward agent has arrived at the source and has updated the probabilities there, it will be deleted.

The probability of the entry corresponding to the node f from which the backward agent has just arrived is increased using the following formula:

$$P_{new,f} = \frac{P_{old,f} + \Delta P}{1 + \Delta P} \quad (3)$$

- $P_{new,f}$ is the new probability;
- $P_{old,f}$ is the old probability;
- ΔP is the probability increase.

ΔP should be inversely proportional to the age of the forward agent. The formula we use is:

$$\Delta P = \frac{A}{t} + B \quad (4)$$

Where A and B are constants and t is the trip-time of the forward agent from this node to the destination node. This trip-time is the sum of the trip-times from this node via the node along the way to the destination node of the forward agent. We do not take into account that the conditions of the traffic network can change from the moment that the node is visited by the forward agent and the updating of the backward agent.

The other entries in the probability table with the same destination but other neighbouring nodes are decreased using the formula:

$$P_{new,i} = \frac{P_{old,i}}{1 + \Delta P}, \quad \forall i \neq f \quad (5)$$

These formulas ensure that the sum of the probabilities per destination node remains 1. Probabilities can only decrease if another probability increases. Probabilities can approach zero if other probabilities are increased much more often. This is not very desirable, because in time it may appear that the choice associated with that probability is the best at that time, but the agents will not detect it because they hardly ever take that route. This problem can be solved after analogy with the natural ants: they do not always use the pheromone trail as their guide, but sometimes just explore new routes. Therefore we introduce an exploration probability as a minimum value for each probability. An example could be 0.05 divided by the number of next nodes. After setting this minimum, the probabilities per destination are normalised to one again. This ensures that none of the entries in the probability table will ever reach zero.

For a given value of ΔP , the absolute and relative increase of P_{new} is much larger for small values of P_{old} than for large values of P_{old} . This results in a weighted change of probabilities. The formulas were taken from [Van der Put 1999].

Very often the probability tables are initialised with equal values in such a way that all the probabilities for one destination sum up to 1. This way the first agents do not have any information about routes, let alone the quality of the routes. The performance of the Routing system will, in that case, be very bad at the beginning and cannot even be evaluated properly, because of routes with cycles. The quality of the routes found by the agents improves with time. At first the agents may find

many cycles, but the number of cycles decreases as the probability tables are filled with information that is more accurate.

We have chosen to initialise the probabilities to values that are biased to the best routes in a static network environment. The probabilities for these routes start with a high value and the other probabilities start with a lower value. This way the ants will start exploring routes with a preference for good routes. And no initialising period is needed before the traffic can benefit from the Routing system. The best routes in a dynamic network environment may very well be different from the best routes in a static network environment. The traffic will cause congestions for example. The agents take care of finding those better routes.

As a final point the vehicles will be routed according to the highest probabilities in the tables. They do not have to explore other routes. They just want the best route.

Updating sub paths

We have added some extra functionality for better performance of the Ant Based Control algorithm. The first is the possibility to update sub paths. In the original algorithm the backward agent steps back from the destination to the source. On its way it updates the probabilities from its current node to the destination. However it also has information about the routes to other nodes on the way. Therefore the possibility is added to let the backward agent also update the probabilities for sub paths of the route of the agent. We illustrate this with the aid of Figure 29.

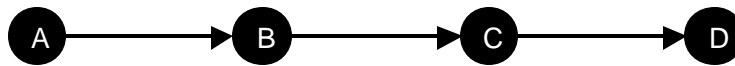


Figure 29: The path of a forward agent

A forward agent moves from its source node A to its destination node D via nodes B and C. According to the Ant Based Control algorithm the backward agent then moves back from D to A via C and B. In the original algorithm the backward agent updates the probabilities for the route from C to D when it is in node C. Arrived at node B the agent updates the probabilities for the route from B to D. And at node A the probabilities for the route from A to D are updated. Summarising the updated routes:

- From C to D
- From B to D
- From A to D

Updating sub paths means that the agent also updates the probabilities for the route from B to C when arrived at node B. And when the agent reaches its source node A the probabilities for the routes from A to C and from A to B are also updated. Summarising the extra updated routes:

- From B to C
- From A to C
- From A to B

For these extra updates no extra information is needed and the agent does not have to follow a longer route. So there are more updates for almost no extra costs.

Using travel time differences

Earlier in this section we discussed how the probabilities are updated. Therefore we used DP from

$\Delta P = \frac{A}{t} + B$. In this formula t stands for the travel time from the current node to the destination node.

The consequence is that the update is small for large values of t and bigger for small values of t . So imaging that an agent has for example two alternatives: a long path and a small path. Taking the long path results in a small update for that path. And taking the small path results in a big update for that small path. This works fine for relatively close nodes: the smaller path is preferred above the long path. But when nodes are further away from each other, two alternative paths are both rather long.

Even when the travel times are quite different, both paths will still receive small updates. Because of these small updates none of the probabilities will offer a clear preference for one path or the other. To overcome this problem somewhat, we can compose a t from two numbers. The first part is the usual travel time; the other part is the experienced delay. For this delay we compute the minimum travel time for a vehicle to follow the same route as the agent did, and we subtract this from the real travel time that the agent measured. We call this the travel time difference: the difference between the measured travel time and the minimum travel time for the same route. This minimum travel time can be computed with the maximum speed for the given links and the length of the links. These two times can be combined in any relation you want. For example, they can be combined fifty-fifty: using 50 % of the real travel time and 50 % of the travel time difference. The combined value is used for variable t . Using travel time differences makes the updates of the probabilities less dependent on the distance between the nodes.

3.4.3 Distributed components

The Routing system is composed of several distributed components. Both the Timetable updating system and the Route finding system are divided over the distributed components. Every component takes care of a part of the traffic network. So every component has the functionality of the Timetable updating system and the Route finding system. Furthermore every component contains the dynamic data for the part of the network concerned. And finally, they all have the same static data. Information required by one component but stored at another component must be communicated.

Communication of Timetable updating system

Some of the vehicles send information about their covered route to the Routing system. This information is handled by the Timetable updating system. This system might consist of several distributed components, but only one of the components receives the information. The information is processed by this component. The result can be one or more measurements for one or more roads. See Equation 1 in section 3.4.1 for the computation of the measurements. These measurements can be designated for roads handled by this Timetable updating system component as well as for roads handled by other Timetable updating system components. In the latter case the measurement will be sent to the appropriate component. At each Timetable updating system component the measurements for some of the roads are collected. These measurements are regularly used in the computation of the weighed average travel time for each road. See Equation 2 in section 3.4.1.

Communication of Route finding system

The Route finding system uses ants to compute the shortest routes with the available information. When the Routing system is distributed over several components, so is the Route finding system. This means that such a component only takes care of some of the nodes in the network. The ants hop from node to node to collect information and compute the shortest routes. Hence it can happen that an ant has to move from one node, handled by one Route finding system component, to another node, handled by another Route finding system component. In that case communication between the components takes care that the ant is moved from one component to the other. There are separate messages for forward ants and backward ants. See section 3.4.2 for a description of the Ant Based Control algorithm. The Route finding system also handles the requests for routes from drivers. Such a request is sent to one of the components. In many cases however, this component cannot provide a complete route because it does not take care of all the nodes of the route. The route is composed of a list of nodes to be followed by the vehicle. The information in each node can provide one step of the route and for every next step the next node must be consulted. So it is very well possible that some parts of the route requested by the driver must be taken from another component. For this purpose similar requests as the one from the driver are sent to other components, but with a different starting point of the route. This way the route will be composed of small parts from several components. When the route is complete or the maximum number of nodes in a route is reached, the response is sent back to the driver.

Chapter 4: Implementation

4.1 Introduction

This chapter is a guide for the reader who wants to know how the design, discussed in the previous chapter, is implemented. It will also provide help to anyone that needs to adjust the resulting program. Furthermore it also contains some design details for anyone that found the previous chapter was too scanty. And last but not least this chapter contains information needed to understand the working of the experimental environment for researchers. The implementation is made in Delphi, using Borland Delphi 5.0 Professional. This development environment was chosen because there was software available as an example of an implementation of the ABC-algorithm. Furthermore this environment makes it easy to implement a user interface that can show a simplified traffic network of a city. In the same way graphics to show the statistics of the performance are easily added. The applications are developed for Windows 2000, although they should work with most other versions of Windows. As a reminder the system design from the previous chapter is repeated below. The figure makes it clear that two applications have to be built: the City program and the Routing system.

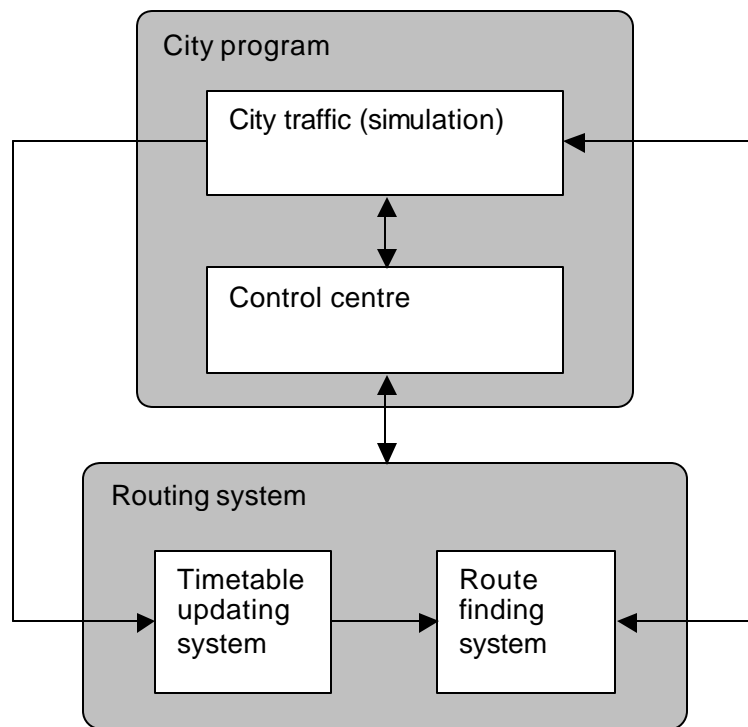


Figure 30: System design

In this chapter the implementation will be discussed in the following order:

1. Traffic simulation
 - Loading a traffic network
 - Changing the traffic network
 - Saving the traffic network
 - Automatic simulation preparation

2. Routing system
 - Loading the agent network
 - Automatic simulation preparation
 - Saving routing tables
3. Running a simulation
 - Manual simulation preparation
 - Running the simulation
 - Evaluating the simulation
4. Special functionality
 - Zooming
 - Integer queues
 - Integer rings

4.2 Traffic simulation

We start with a discussion of the traffic simulation. Below is a sample of a screenshot of the program where the traffic is simulated. Displayed is a network of streets with vehicles as moving bullets. It may look like there is depth in this view, but that is actually the result of fitting the network in the window. The network is stretched in width to fit, and resizing the window would also rescale the network.

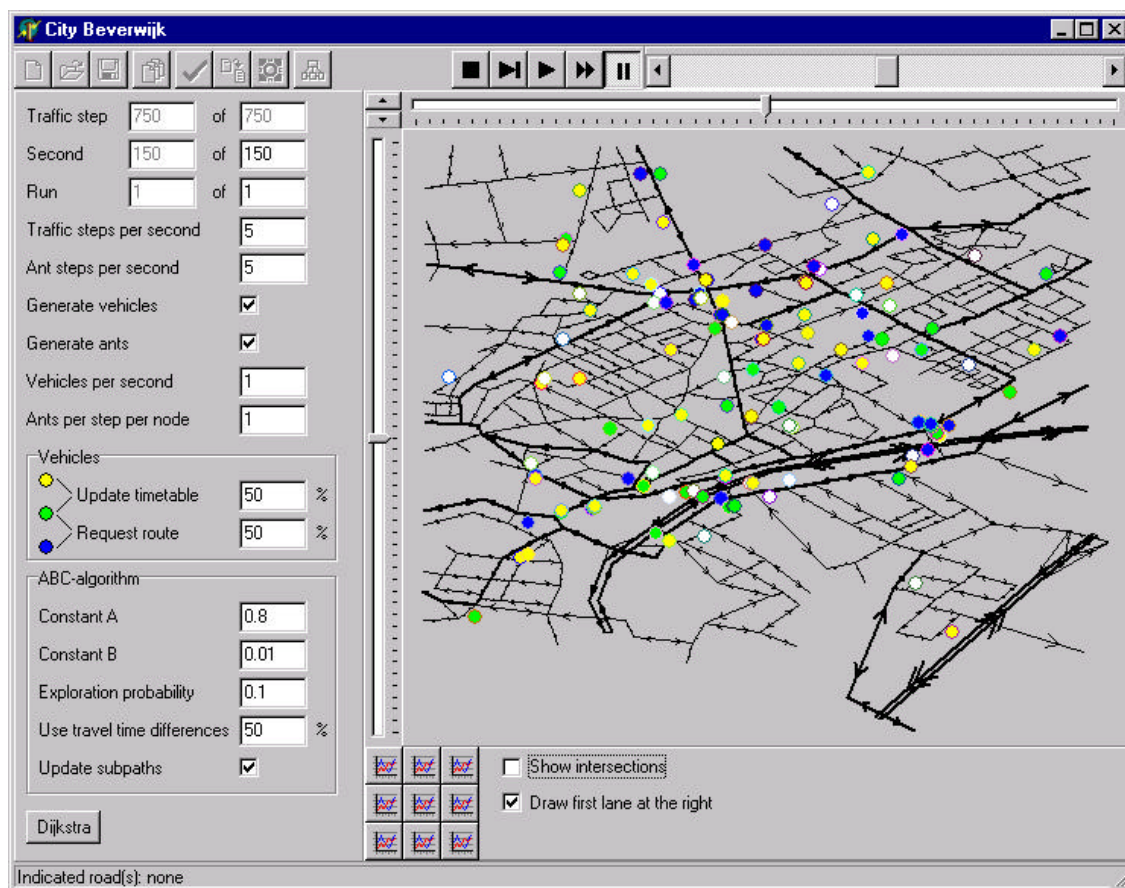


Figure 31: Example of a traffic network

4.2.1 Loading a traffic network

To create a traffic network as shown in Figure 31 a lot of data is needed, such as the characteristics of roads and intersections, capacities etcetera. This data will be stored in several data files. This way the network can easily be adapted to changes. Now the first implemented module takes care of reading this information from files. Without any information there is nothing to visualise or to route.

To make a realistic simulation of the traffic network the following data are needed in the programs:

- *A network of roads and intersections:*
We will call the intersections 'nodes' and the roads are called 'links'. Nodes are typically the points where roads split and a driver can choose between several directions to continue his way. For every node we need the position on the map. This position consists of a nonnegative x- and y-coordinate. For a link we must know which nodes it connects and in what direction, because the links are directed. So we need the node where the link starts from and the node where the link ends. In most cases there will be two links between two nodes, because the traffic can drive in both directions. For this and other purposes the nodes and links have to be identifiable. So every node and every link gets a number. These data are both required by the City program and the Routing system. The data is stored in files with the extension '.map'.
- *Intersection properties:*
For every node we need to know the type. This can be a normal intersection, an intersection with traffic lights or a roundabout. This information will be used in both programs. The data is stored in '.int' files.
- *Road properties:*
For every link we want to know the number of lanes, the length, the average speed and the fact whether or not it is a main road. Because the priority of a road can change half way, it is important that we only mark a road to be a main road, when it has precedence at the intersection where it is going at. So a road, going from an intersection where it has precedence to an intersection where it does not, will be marked with a low priority. Only the City program needs the number of lanes and the priority of the road. The length and speed will be used in both programs. This information is stored in '.road' files.
- *Routing distribution information:*
For every node we need to know which part of the Routing system will take care of it. For every number of Routing system part that is found, there will be a Routing system set up. The Routing systems will deal with the updates that the vehicles send about their position and with the request for routes from the vehicles. This information is needed both by the City program and the Routing system and will be stored in '.dis' files.
- *Traffic light tables:*
For every node of type 'traffic lights' we need a traffic light table. As mentioned in the previous chapter, a traffic light table describes for every road that ends at the intersection at what time the light is green. These times are used periodically. Therefore a cycle time is needed for every intersection with traffic lights. Such a cycle time can for example be 60 seconds. This would mean that every 60 seconds the time for the traffic light is set to 0. Now we identify each road to this intersection by the number of the intersection where the road starts. And for every 'from' node and every possible direction (left, right or ahead) the start and end time of the green light are stated. Only the City program needs this information. It will be stored in '.light' files.
- *Default routing tables:*
Every node gets a default routing table. In that table all possible destination nodes are summed up, that is, all nodes except the node for which the table is destined. And for all these destinations a neighbouring node of the current node is set to be the best choice to go. This information will be used by the vehicles of the City program. The Routing system will use this information for an initial state of the probability tables and to be able to count the difference between the default routing tables and the current settings of the Routing system for statistical purposes. The information will be stored in '.route' files.

- *Source and destination rates:*
For every link we need to know a source and destination rate. A source rate determines how many vehicles will start at the link relative to how much vehicles will start at the other links. For example there could be three links: link 1 has a source rate of 100, link 2 has a rate of 200 and link 3 has a rate of 0. Then the relative source rates are 100/300, 200/300 and 0/300 respectively. So 33 % of the vehicles will start at link 1, 67 % of the vehicles will start at link 2 and no vehicles will start at link 3. The destination rate determines how many vehicles will choose the link as their destination. Only the City program will use this information.
- *City environment:*
In one file all the files that belong together are stated. This is done in a file with a '.city' extension. In this file seven variables are assigned a value that determines which files are used together. This is the main file. Loading this file will automatically load the appropriate files of a city environment. The extension of the file is '.city'.

File structure

The module Script makes it possible to read files that are written in a C-like style (slightly different from the Delphi-style, the used programming language). This style enables the information files to be structured and supplemented with comments. In this style white space characters like spaces and tabs are ignored, just as comments following '/' or comments between '/*' and '*/'. The data files for the City program are text files that can be edited with any ordinary text editor. The necessary structure is made clear by giving an example of the files below. The sample files are supplemented with comment. Comment is skipped when a file is read. The sample files would result in a network with three nodes connected as a triangle with six links (Figure 40).

<pre>// Use this file to point out the files // needed to run a simulation // The required files are: /* map_file intersection_file road_file distribution_file light_file route_file rate_file */ map_file = "Sample.map" intersection_file = "Sample.int" road_file = "Sample.road" distribution_file = "Sample.dis" light_file = "Sample.light" route_file = "Sample.route" rate_file = "Sample.rate"</pre>	<pre>// Use this file to construct the map // First enumerate the intersections // Then connect the intersections with // roads intersections { // {number, x-pos, y-pos} {1, 50, 0} {2, 100, 100} {3, 0, 100} } roads { // {number, from, to} {1, 1, 2} {2, 2, 1} {3, 2, 3} {4, 3, 2} {5, 3, 1} {6, 1, 3} }</pre>
--	---

Figure 32: Sample.city

Figure 33: Sample.map


```
// Use this file to assign a type to the
// intersections

// Possible types:
// NORMAL
// TRAFFIC_LIGHTS
// ROUNDABOUT

intersection_types
{
    //{number, type}
    {1, NORMAL}
    {2, TRAFFIC_LIGHTS}
    {3, ROUNDABOUT}
}
```

Figure 34: Sample.int

```
// Use this file to assign the road
// properties

// the properties are:
// a number of lanes
// a length (in m)
// a speed (in km/h)
// a priority (HIGH or LOW)

road_properties
{
    //{road number, number of lanes,
    //length, speed, priority}
    {1, 1, 200, 50, LOW}
    {2, 1, 200, 50, LOW}
    {3, 2, 250, 45, LOW}
    {4, 2, 250, 45, LOW}
    {5, 1, 260, 55, HIGH}
    {6, 1, 260, 55, HIGH}
}
```

Figure 35: Sample.road

```
// Use this file to distribute the
// intersections over the separate parts of
// the routing system

intersection_distribution
{
    //{intersection number,
    //routing system part number}
    {1, 1}
    {2, 2}
    {3, 2}
}
```

Figure 36: Sample.dis

```
// Use this file to assign the traffic
// light tables to the intersections
// controlled by traffic lights

// Use only the intersections that are
// assigned the type TRAFFIC_LIGHTS
// The cycle time is the time in seconds
// before the cycle starts again
// Use only those 'from' intersections that
// have a link to the concerning
// intersection

// Possible directions:
// LEFT
// AHEAD
// RIGHT

// start and end time of green in seconds

traffic_light_tables
{
    //{intersection number, cycle time
    {2, 60,
        //{ 'from' intersection,
        //to direction,
        //start time of green,
        //end time of green}
        {1, RIGHT, 0, 25}
        {3, LEFT, 30, 55}
    }
}
```

Figure 37: Sample.light

```

// Use this file to make default routing
// tables

// Enumerate all intersections
// For all intersections, enumerate all
// other intersections as destinations
// (so the intersection itself is no
// destination)
// Assign each destination a next
// intersection
// This next intersection will be the
// choice of direction for the unguided
// vehicles
// This next intersection must of course be
// directly connected from the source
// intersection

default_routing_tables
{
    //intersection number
    {1,
        //{destination,
        //next intersection}
        {2, 2}
        {3, 3}
    }
    //intersection number
    {2,
        //{destination,
        //next intersection}
        {1, 1}
        {3, 3}
    }
    //intersection number
    {3,
        //{destination,
        //next intersection}
        {1, 1}
        {2, 2}
    }
}

```

Figure 38: Sample.route

```

// Use this file to assign a load rate to
// the roads

// The source rate determines how much the
// road is chosen as the source of a trip
// The destination rate determines how much
// the road is chosen as the destination of
// a trip

// (source rate per road) divided by
// (sum of source rates) determines
// the percentage for every road

// The same goes for the destination rates

load_rates
{
    //{road number, source rate,
    //destination rate}
    {1, 200, 175}
    {2, 200, 175}
    {3, 150, 25}
    {4, 50, 50}
    {5, 0, 75}
    {6, 0, 100}
}

```

Figure 39: Sample.rate

Loading this environment into the City program would result in the following screenshot.

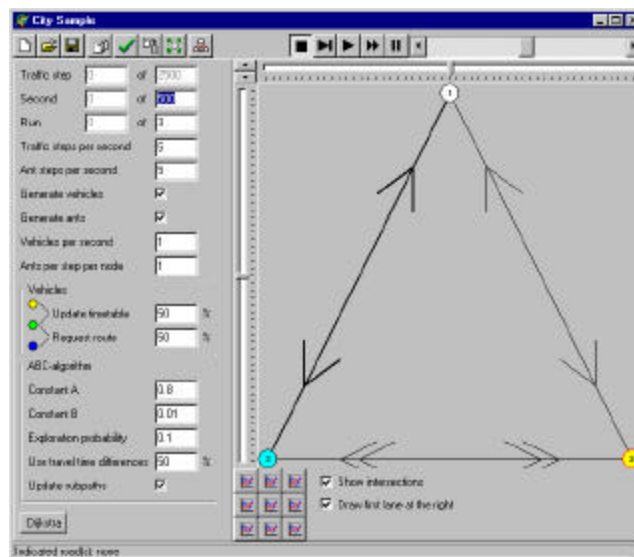


Figure 40: Screenshot of the City program with the sample environment loaded

4.2.2 Changing the traffic network

Changing the traffic network can be done in two ways. The first way is to edit the files with a text editor program like Notepad or Wordpad for example. The other way is to load the environment into the City program. Then the map will be shown. Clicking on a node or a link will pop up a form where the properties can be changed. Saving the environment will change the text files. A few things cannot be changed from within the program. These things are adding and removing of nodes and links, and changing the 'from' and 'to' node of a link. These things have to be done by changing the '.map' file with a text editor program.

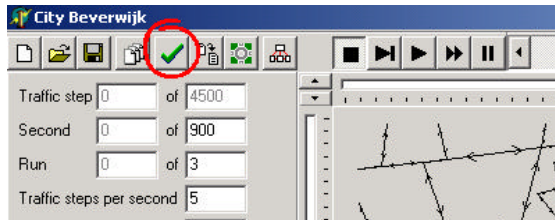


Figure 41: Check if data complete

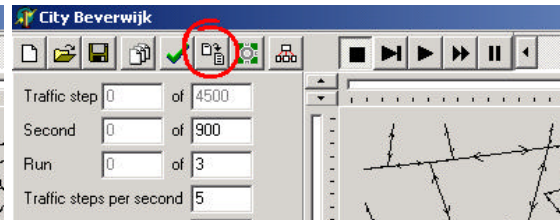


Figure 42: Supplement missing data

Use the green check button to check if the environment information is complete. All nodes and links are checked until one of them is incomplete. Missing data can be added as described in the former paragraph, but there is also a button in the program to automatically supplement data where missing. For most properties a default value is used. But for the length there is a scaling factor. When a link has no length yet, the distance between the two nodes it connects is computed by using the node coordinates and the acquired length is multiplied by the default scaling factor. Adding a default traffic light table is done with the aid of a default cycle time and a default evacuation time. The cycle time is divided over the incoming links and the evacuation time is subtracted from the time per link. Suppose the default cycle time is 60 seconds, the default cycle time is 5 second and there are 4 incoming links. Then every link gets $((60 / 4) - 5 =) 10$ seconds green light. So the first incoming link will have a green light from 0 to 10 seconds, the second incoming link will have a green light from 15 to 20 seconds, etc. The gap between the first and second link is the evacuation time. The green times are by default set to the same values for all outgoing directions (left/right/ahead). When there is no default route entry for a certain destination the first outgoing link is added. This will mostly not be the best choice and it is very likely that the vehicles will not reach their destination at all when using this default value. Therefore there is also a button named Dijkstra. When pushing this button Dijkstra's algorithm for shortest paths is used to compute the shortest paths from all (source) nodes to all (destination) nodes. The algorithm uses the length and speed properties of the links to compute the travel time per link. These travel times are used to determine the shortest paths. See section 2.1.1 for a description of Dijkstra's algorithm.

4.2.3 Saving the traffic network

Saving the traffic network can be done by pushing the save button. There is also a button to change the filenames. To save files with a different name than the loaded files, first change the name(s) and then push the save button. When saving the files the comment from the loaded files is skipped. Instead of that the standard comment is added to the files. Take this into account when adding your own comment to the files: it is replaced with the standard comment when saving the files from the City program. It is recommended not to save an environment after transforming the roundabouts (see next section). It is not easy to change the situation back to the roundabout, because the original node is removed, new nodes are created, links are changed and added and all default routing tables are adjusted. There is no need to save the transformation of the roundabouts unless the default

transformation settings do not satisfy the requirements. See the next section for a description of the changes made when transforming a roundabout.

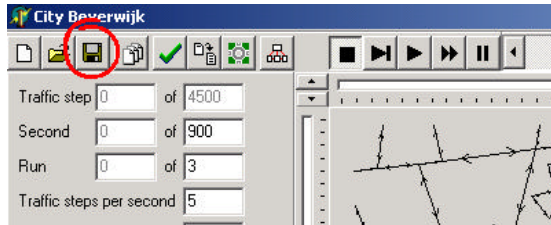


Figure 43: Save this city

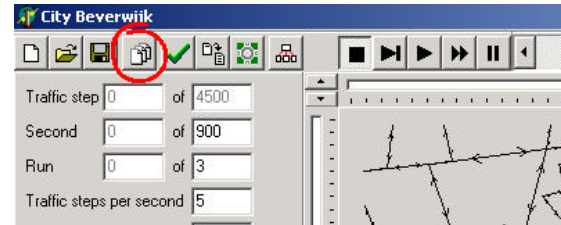


Figure 44: File names

4.2.4 Automatic simulation preparation

Transforming roundabouts

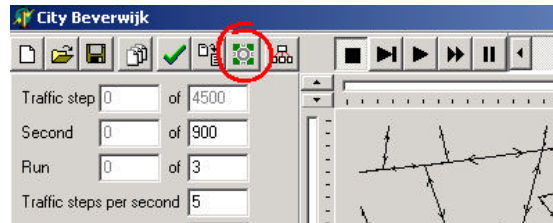


Figure 45: Transform roundabouts

Before a simulation can be started a few things have to be done. The first thing is to push the button to transform the roundabouts (if any). All roundabout nodes in the network (blue) are transformed to sets of normal nodes (white). This process includes a number of steps that are not directly visible.

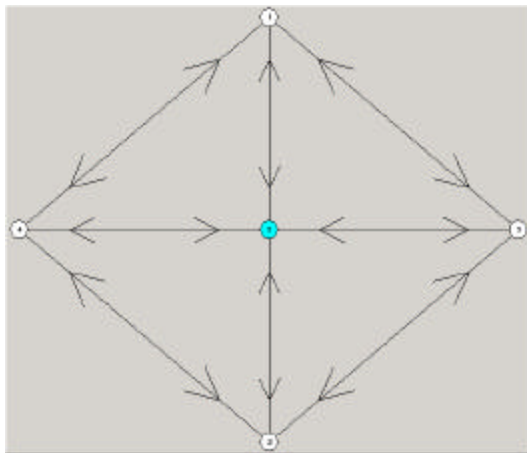


Figure 46: Map with untransformed roundabout

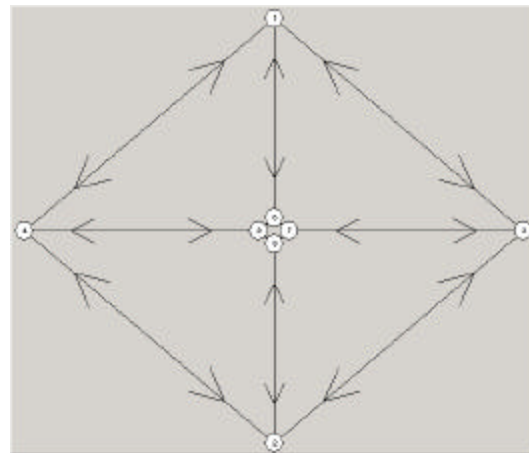


Figure 47: Map after roundabout transformation

For every link that approaches the roundabout node a new node is created. Also a new node is created for every link that leaves the roundabout but is going to another node than the approaching nodes. Simply spoken, this means there will be one new node for every one-way link and one for two bi-

directional links. These new nodes are positioned on the links a little away from the original roundabout node, let us say 5 meter. The Routing system numbers for the new nodes are inherited from the original node. The links that were connected to the original node are then connected to the appropriate new nodes. When there is a link from the new nodes going to a node with traffic lights, the node numbers in the traffic light table are changed from the original node number to the new node number. The new nodes also have to be connected to each other. This is done by comparing the relative positions of the new nodes. This makes it possible to create a circle of new nodes and new links, where the links are directed counter clockwise. The new links get a default length and speed, one lane and a source and destination rate of zero. The nodes around the composed roundabout that have default routing tables pointing to the original node are adjusted to point to the appropriate new node. The default routing tables of the new nodes are inherited from the original node where possible and for all other destinations traffic is routed around the roundabout. The default routing tables of all nodes are extended with the new nodes as possible destinations, where the next nodes in the tables are copied from the original node. All these steps together will be done after pushing the button to transform roundabouts. There is no automatic reversing procedure, so it is recommended not to save the environment after transforming the roundabouts. This way it is easier to change the intersection type from roundabout to something else.

Direction determination

In the previous chapter we discussed intersections controlled by traffic lights. For every road at any intersection with traffic lights there can be three sets of traffic lights: one for going left, one for going ahead and one for going right. When a vehicle wants to go from one road approaching an intersection to another one that leaves the intersection, it needs to know what direction it is going (left, right or ahead). This information is not directly available, but derived from the geometry. The computation is done automatically after every change in the network. In the simplest case the vehicle has only one possible road to go to (going back not taken into account). Then the name of the direction can be derived from the angle that the vehicle will make. This will be explained with Figure 48.

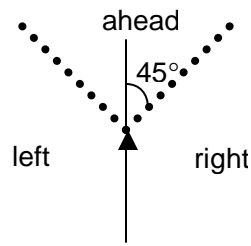


Figure 48: The name of the direction depends on the angle of the roads

When the angle is 45 degrees or less the direction is called 'ahead'. When the angle is larger than 45 degrees the direction is called 'left' or 'right' according to the side the road is on. The case, when the vehicle has two possible roads to go, is a little more complex. This can most easily be described by summing up all possible situations with two roads. All situations with two roads can be categorised in one of the following templates, according to section the roads are in.

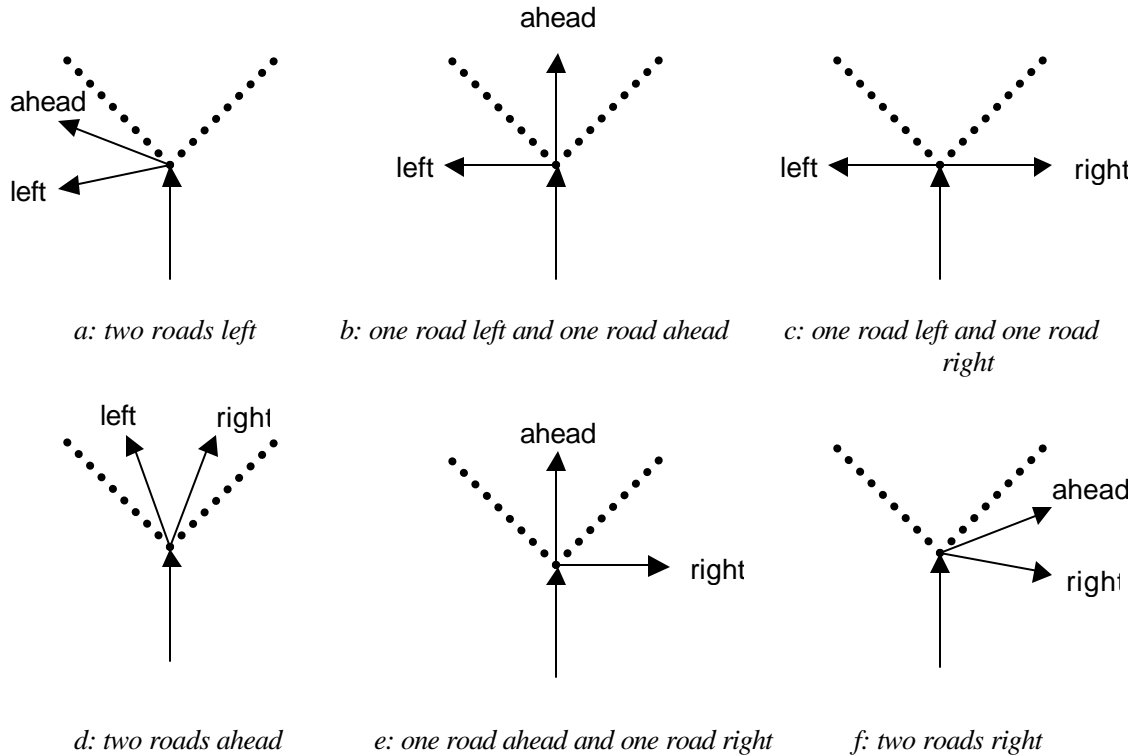


Figure 49: All possible situations with two roads to go to

This way of labelling, results in most cases to the most intuitive names for the directions. But there might still be some instances where people would choose another label. The cases where the vehicle has more than two choices to go to are much easier classified. The leftmost road is called 'left', the rightmost road is called 'right' and all other roads in between are called 'ahead'. This is certainly not always the best way to label the directions, but it makes the system more predictable. When a complex algorithm would be used to label the directions, the system would be badly understood by the user and might do unexpected things the user cannot explain. One must notice that the backward road is always left out when counting the number of possible roads to go.

Creating roadblocks

Another thing that is done automatically after changes in the network is the (re)creation of roadblocks. Every link is divided into a number of roadblocks. Every roadblock can at most contain one vehicle. This is used to avoid vehicles bumping onto each other and keep a distance between vehicles. The number of roadblocks per link depends on the length of the link and the number of lanes. There is a roadblock for every 10 meter (for example) and for every lane. So a link of 25 meter and two lanes will have 3 roadblocks on both lanes, which makes 6 roadblocks. Every time a length of a link or the number of lanes of a link changes, the number of roadblocks is computed again.

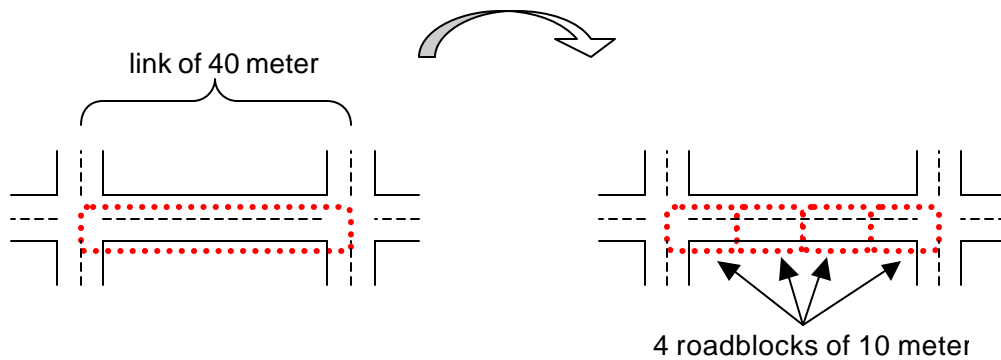


Figure 50: Creating roadblocks on a link

Setting up a Routing system

Some vehicles in the traffic simulation of the City program will use a static default route. Others will use the Routing system for a dynamic route. This Routing system will use the current state of the network to determine the best routes for the vehicles. The Routing system is a separate program or a group of cooperating programs. It can be started from the City program. The first thing to be done for the set up of the Routing system is the inventory of the Routing system parts used by the nodes. When there are for example 4 different numbers of Routing system parts then there will also be 4 different port numbers that identify the different parts of the Routing system. These port numbers will be saved in a temporary file. Then (in this case) 4 Routing system programs are started. These programs will start minimised, so they will be visible in the taskbar. They read their port number from the temporary file and open a connection at the given port. The other port numbers are also read to be able to connect to each other. In the city program the port numbers are stored at every node. This way information can be sent from a node in the city program to the appropriate Routing system part. Finally the Routing system programs connect to the city program to acknowledge their readiness. Now the simulation can be started. Setting up a Routing system is done automatically after pushing a button. See Figure 51 and Figure 52.

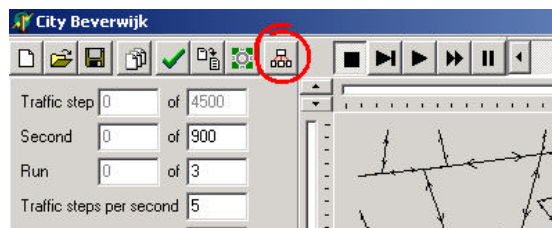


Figure 51: Start distributed routing system



Figure 52: The taskbar shows the City program and four Routing system parts

4.3 Routing system

The traffic from the traffic simulation is partially routed by fixed routing tables in the City program. Drivers follow this route because of their knowledge of the environment or because they follow the advice of a static route planner. Another part of the traffic uses the Routing system to find their route based on dynamic data. This Routing system maintains dynamic routing tables. These routing tables are adjusted to the current traffic in the traffic network. This means for example that the routing tables

are adapted to route traffic via quiet roads when these are faster than the busy roads, which might be faster at quiet times. In general the traffic is routed via paths, which are not necessarily the shortest in distance, but the shortest in time. Another important aspect of the Routing system is the distributed handling of the problem. The Routing system is prepared to run on several computer systems at once. The different parts of the Routing system will cooperate by communication through TCP/IP. Each part of the Routing system takes care of a portion of the traffic network. This will increase computer speed and space for better performance of the system. Opening one or more Routing system programs is done automatically from the City program by pushing a button as described earlier in this chapter. Opening a Routing system program directly will terminate the program because of improper command line parameters. The Routing system program is not intended to be opened otherwise than from the City program.

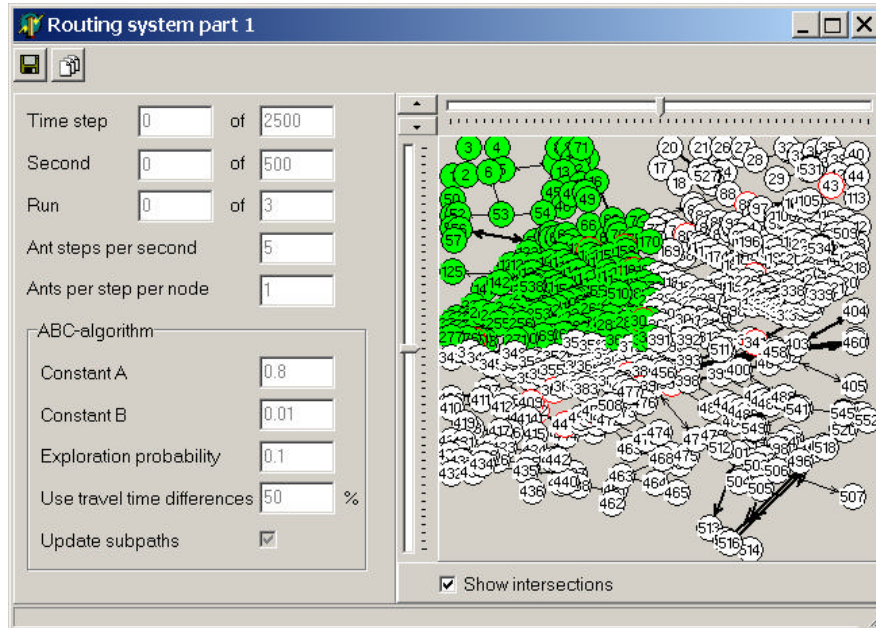


Figure 53: An example of a Routing system program

4.3.1 Loading the agent network

When the Routing system program is started it automatically tries to load a network for the agents. For this purpose it evaluates the command line parameters, which were set by the City program. A subset of the files that the City program used is now loaded into the Routing system program. First of all the '.city' file is loaded to read the location of the other files. Then the '.map' file is read to construct the network topology for the agents. Also the '.int' file is read in which the intersection types are stored. These data are read for the purpose of identifying the roundabouts. As transforming roundabouts changes the network topology. The '.road' file with the road properties is read. From this file the number of lanes and the priority per road are read only for visualisation purposes. The number of lanes determines the number of arrows per road, and the priority determines the line width. These values do not contribute to the operation of the program. The length and speed per road do contribute to the operation of the program. The length is used for travel time estimations per road. The speed is used for default travel times. The distribution file is read as well. In this file is stated which intersections are handled by which part of the Routing system. This information is needed to be able to communicate the right information to the right Routing system program. In Figure 53 the intersections handled by that part of the Routing system are coloured green. The '.light' file is not read, because the Routing system will not use the information that controls the traffic lights. The '.route' file, which contains the information of the default routes, is read. These default routes can have been entered manually or computed by Dijkstra's algorithm for shortest paths. This information is used to initialise the

probability tables. The probability tables could also be initialised with equal probabilities for all possible route options. In that case however, it may take some time to find the best probabilities for optimal routes. To avoid that problem the default routes are given a slight advantage with regard to other routes, assuming that the default routes are a considerably good starting point. In addition the default routing tables are used to compare the results of the probability tables, so we can show the differences. Finally the '.rate' file, which contains information about the traffic distribution over the network, is skipped because it contains no information for the Routing system.

4.3.2 Automatic simulation preparation

The first thing that is done automatically after loading the agent network is the transformation of the roundabout nodes. This procedure is the same as the transformation of the roundabouts in the City program described earlier in this chapter, except for one thing. The Routing system program does not use traffic light control information, so there are no traffic light tables adjusted in the Routing system program. For the extended description of the procedure to transform the roundabouts, see 'Automatic simulation preparation' within the City program. Another automatic action is the creation and initialisation of the probability tables. As mentioned earlier the probability tables are initialised with a slight precedence for the default routes. As a last thing the connection with the City program is established and the Routing system program receives the parameters for the simulation from the City program.

4.3.3 Saving routing tables

The save button in the Routing system can be used to save the routing tables. Every node in the Routing system has a probability table, which is used for routing the traffic. In such a table only the highest probabilities are used for the traffic: the nodes belonging to the highest probabilities determine the routes provided to the drivers. Saving this information will result in a '.route' -file like 'Sample.route' in section 4.2.1. The name can be entered in a dialog. In this file not the probabilities are saved, but the current best choices for the vehicles. This way the file will contain the default routing tables that could be used to route the standard vehicles: the vehicles that do not use the Routing system to be guided. In case that the Routing system is distributed over several Routing system parts, not all the probability tables are available in one Routing system part. So when saving the information from the probability tables in one part, not all routing tables can be made. And thus one has to save the routing tables from all Routing system parts to acquire all routing tables. This information can then be combined in one file, by smart copying and pasting with a text editor.

4.4 Running a simulation

When the data is complete, the roundabouts are transformed and the Routing system is started, a simulation can be started. This can be done by simply pushing the play button. But there might be a few settings to be adjusted before starting.

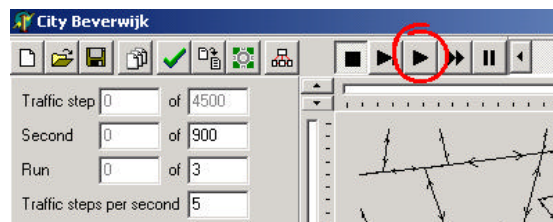


Figure 54: Run simulation

4.4.1 Manual simulation preparation

The City program shows a list of parameters that can be adjusted before starting the simulation. These parameters will be discussed one by one.

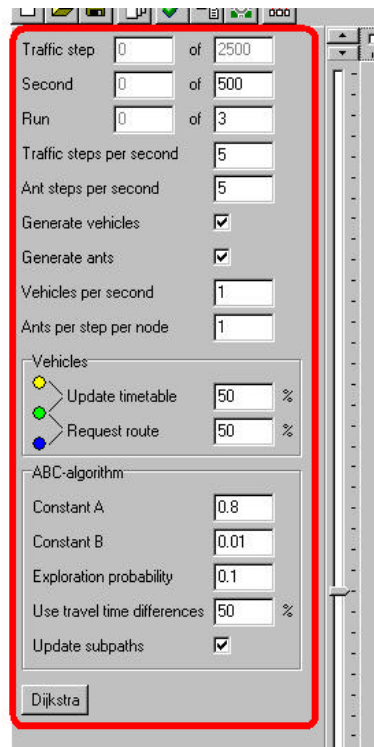


Figure 55: The parameters of the simulation

- **Traffic step:** the number of steps that the traffic in the simulation makes. The first box shows the number of steps already done. The second box shows the maximum number of steps per run of a simulation. This value is the multiplication of the parameters 'Seconds' and 'Traffic steps per second'.
- **Second:** the number of seconds that a run of a simulation takes. The first box shows the number of seconds passed. The second box shows the maximum number of seconds per run of a simulation. This value can be adjusted in a range from 0 to 999,999.
- **Run:** the number of runs of a simulation. The simulation can be run a number of times without a pause. The results of different runs will be different, because a randomiser is used. The first box shows the current run. The second box shows the maximum number of runs. This value can be adjusted in a range from 0 to 999,999.
- **Traffic steps per second:** the number of steps that the vehicles will make per second. This value can range from 0 to 99. Setting this value to low will result in an inaccurate traffic simulation. Setting this value to high will make an unnecessarily high demand on the computer system.
- **Ant steps per second:** the number of steps that the ants will make per second. This value can range from 0 to 99. The ants will hop from one node to another every step. So this value determines the speed of the ants.
- **Generate vehicles:** whether the simulation should run with or without vehicles. In most cases it would be useless to run a simulation without vehicles.

- *Generate ants*: whether the Routing system should generate ants. Not generating ants would make it useless for vehicles to update the timetable or request a route from the Routing system (see concerning parameters), because the probability tables will not be updated.
- *Vehicles per second*: the number of new vehicles per second. At the beginning of a run of a simulation there is no traffic at all. This parameter controls the number of vehicles that will be added per second. The value can range from 0.00 to 9,999.
- *Ants per steps per node*: the number of new ants generated at every node and every step. The value can range from 0.00 to 9,999.

Vehicles:

- *Update timetable*: the percentage of the vehicles that will send update information to the timetable of the Routing system. This value can range from 0 to 100.
- *Request route*: the percentage of the vehicles that will request a route from the Routing system. This value can range from 0 to 100.

ABC-algorithm:

- *Constant A*: a parameter used in the algorithm to update the probability tables. The value can range from 0.0000 to 999,999. See the description of the ABC-algorithm for further details.
- *Constant B*: a parameter used in the algorithm to update the probability tables. The value can range from 0.0000 to 999,999. See the description of the ABC-algorithm for further details.
- *Exploration probability*: a parameter used in the algorithm to update the probability tables. The value can range from 0.0000 to 999,999. See the description of the ABC-algorithm for further details.
- *Use travel time differences*: instead of the real travel time of the vehicles the ABC-algorithm can also use the difference between the real travel time and the minimum travel time. Use this parameter to define how the time is composed from these two options. The value can range from 0 to 100.
- *Update subpaths*: check this option when the agents should not only update the route to their destination, but also the route to all other intersections on their way.

The button at the bottom is not actually a parameter:

- *Dijkstra*: pushing this button will start the computation of the shortest paths with the aid of Dijkstra's algorithm. The algorithm uses the length and speed per road to compute the time per road. The outcome corresponds to the one of a static routing system. It does not take into account any possible delays. The results will be stored at the routing tables of the intersections in the traffic network. This way all vehicles that do not use the Routing system will follow the resulting routes of this computation.

A number of these parameters is also (or only) used by the Routing system. These parameters cannot be changed from a Routing system program. They can be changed in the City program and will then be sent to all parts of the Routing system automatically. This way all parameters can be changed from one view, while it affects all parts of the Routing system at once.

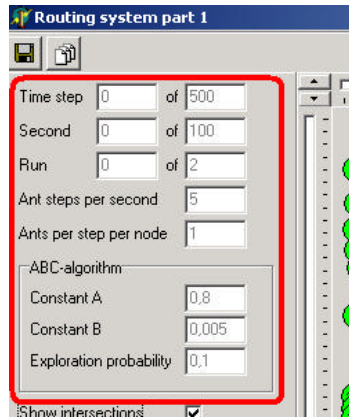


Figure 56: The parameters of the Routing system

After the simulation is started the first eight buttons cannot be used any longer. To be able to use these buttons again the simulation must be stopped and reset by pushing the stop button.

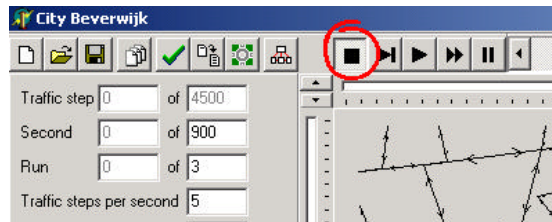


Figure 57: Stop and reset simulation

4.4.2 Running the simulation

Now we will discuss what happens during a step of the simulation. This is explained with the aid of Figure 58. All initiatives start at the Central controller in the City program. First the Central controller (1) signals the Traffic simulation to do a time step. This will make the traffic move. Then the Timetable updating system is instructed to (2) compute the travel times for all roads (again). And finally (3) the Route finding system is stimulated to find new best routes by letting the ants hop from node to node. Thereby they re-compute the probabilities in the probability tables. These steps will now be explained in further detail.

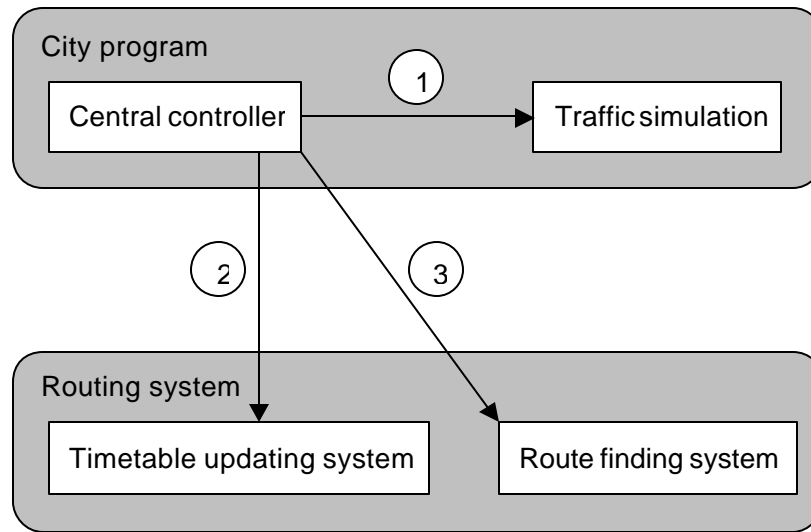


Figure 58: The central controller commands the Traffic simulation, the Timetable updating system and the Route finding system

1. Time step of the Traffic simulation

Before the traffic can be moved two things have to be done. The state of all traffic lights has to be computed. And there will be a check to detect the occurrence of deadlocks. The computation of the traffic lights is described in section 3.3.2 The deadlock detection will now be discussed in more detail.

Deadlock detection and solving

Imagine for example an intersection with four equivalent roads. When at the same time four vehicles are approaching the intersection from all four roads and they all want to go ahead, then they all need to give precedence to the vehicle from the right. The fact that all vehicles are waiting for each other and there is no more progress in the movement of the vehicles is called a deadlock. In real life such situations do not occur, because people foresee such situations automatically and avoid unnecessary waiting. In that case the first or the boldest driver would take precedence without really having precedence. This needs to be detected in the simulation too.

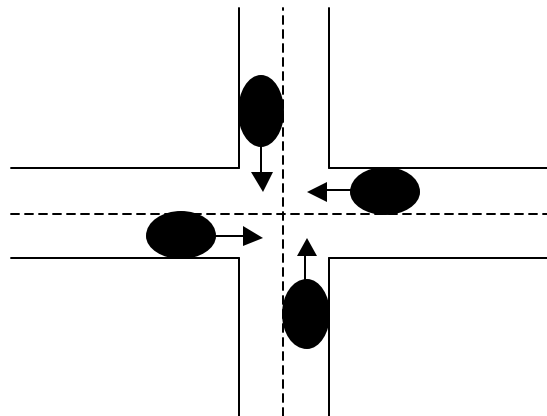


Figure 59: A deadlock, all vehicles are waiting for each other

First of all this is done by registering per intersection whether there are vehicles waiting for precedence this time step. Secondly it is registered per intersection whether there has been a vehicle

this time step that did not have to wait for precedence to cross the intersection. When there is no traffic waiting for precedence or there has been a vehicle that was allowed to cross the intersection there is certainly no deadlock detected yet. When there is traffic waiting and no vehicle crossed the intersection, it is still not certain that there is a deadlock. A vehicle might be waiting for another vehicle that has almost reached the intersection but has not yet crossed it. So another check looks whether there is no traffic at all that has almost arrived at the intersection. In that case a deadlock is detected and can be solved. Therefore, in the next time step, the first vehicle to compute whether it has precedence will get precedence, no matter what other traffic is approaching. This will keep the vehicles driving and solve the appearance of deadlocks. There is however another possibility for the occurrence of deadlocks: the circular deadlock. This occurs when roads are fully occupied with vehicles and some of those saturated roads form a chain. This problem is not dealt with, but it may only occur in heavily loaded networks.

Vehicle actions

The movement of the vehicles consists of three actions. First new vehicles have to be generated with an apparent random source and destination, according to the distribution rates of the links. These new vehicles cannot be placed on a link without checking whether there is space on the link. Therefore each link has a list with vehicles waiting to enter the link. The newly generated vehicles are placed on that (invisible) list. The next step is to place the vehicles from that list on the middle of the links. Before placing the vehicle it is checked whether there is space. If one of the lanes is free at the middle of the link, the vehicle will be placed. Otherwise the vehicle will stay in the list until the next step. The lists have an unlimited capacity to store vehicles. The vehicles in a list will not be visible on the screen, until they are positioned on the link and removed from the list. The final step for the vehicles is the shift over the links. The order in which this is done, is the same order as the link numbers. Link 1 is handled first, then link 2, etcetera. Per link we start with the vehicles at the beginning of the link. For movement rules the vehicles make use of the procedure 'Vehicle.Move'. A simplified version of that procedure is given in pseudo code on the next page.

The procedure for moving the vehicles starts with computing which distance can be covered according to the steps per second and current speed at line 6. Then it gets into a loop that does not end until there is no more distance to cover this turn. The first question in the loop is whether the vehicle can move out of the block it is currently in (line 15). If the vehicle stays in the block it can just move without bothering about other vehicles. When the vehicle moves out of the block the next question is whether the next block is on the same link or on the next link (line 26). When staying on the same link the only question left before moving is whether there is space in the next block (line 29). If there is space the vehicle will be moved into the beginning of that block. Otherwise the vehicle will stop moving this turn and the loop is exited (line 39). When the vehicle wants to move to another link, we must first determine what the next link of the route of the vehicle will be (line 45). Then the question is whether the vehicle can go to the next link (line 47). This depends on (1) whether there is a green traffic light (or no light at all), (2) whether the vehicle has precedence to all other approaching vehicles and finally (3) whether there is space on the next link (no other vehicle standing still). When all these conditions are met, the vehicle moves to the beginning of the next link (line 50-52). Otherwise the vehicle will stop moving this turn and the loop is exited (line 66). When the vehicle moves to the next link, the remaining distance to move might need to be adjusted, because another speed is allowed at that link (line 54-59). The loop is repeated until there is no more distance to be covered left (or the vehicle cannot move any further).

```

1  // This procedure moves the vehicle along its route as much as possible
2  // in the given time step.
3  procedure Vehicle.Move()
4  begin
5      // Determine the maximum distance that can be covered:
6      MoveDistance := CurrentLink.Speed / TimeStepsPerSecond
7      // Try to move as long as there is some distance left:
8      while (MoveDistance > 0) do
9          begin
10             // Determine the current block:
11             BlockNr := GetBlockNr(Position, CurrentLink.Length)
12             // Determine the last position in this block:
13             BlockEndPos := GetLastPosOfBlock(BlockNr, CurrentLink.Length)
14             // Is there not enough distance left to move out of this block?
15             if ((Position + MoveDistance) <= BlockEndPos) then
16                 begin
17                     // Then just move all available distance:
18                     Position := Position + MoveDistance
19                     MoveDistance := 0
20                 end
21             else begin
22                 // Otherwise first move to the end of the block:
23                 MoveDistance := MoveDistance - (BlockEndPos - Position)
24                 Position := BlockEndPos
25                 // Is the next block still on this link?
26                 if ((BlockNr + 1) < CurrentLink.NumBlocks) then
27                     begin
28                         // Is it possible to hop to the next block?
29                         if (CanHopTo(CurrentLink, (BlockNr + 1))) then
30                             begin
31                                 // Move to the first position of the next block:
32                                 Position := Position + 1
33                                 MoveDistance := MoveDistance - 1
34                                 DoHopFromTo(CurrentLink, BlockNr, CurrentLink, (BlockNr + 1))
35                             end
36                         else begin
37                             // If the vehicle cannot move any further,
38                             // set the remaining distance to zero:
39                             MoveDistance := 0
40                         end
41                     end
42                 // The next block is on another link:
43             else begin
44                 // Get the next link on the route:
45                 NextLink := Route.GetNextLink(CurrentLink)
46                 // Is it possible to hop to the next link:
47                 if (CanHopTo(NextLink, 0)) then
48                     begin
49                         // Move to the first position of the next link:
50                         Position := 0
51                         MoveDistance := MoveDistance - 1
52                         DoHopFromTo(CurrentLink, BlockNr, NextLink, 0)
53                         // Is the speed on the next link different:
54                         if ((MoveDistance > 0) and (CurrentLink.Speed <> NextLink.Speed) then
55                             begin
56                                 // Change the remaining distance to move:
57                                 TimeLeft := MoveDistance / CurrentLink.Speed
58                                 MoveDistance := TimeLeft * NextLink.Speed
59                             end
60                         // The vehicle is now on the next link:
61                         CurrentLink := NextLink
62                     end
63                 else begin
64                     // If the vehicle cannot move any further,
65                     // set the remaining distance to zero:
66                     MoveDistance := 0
67                 end
68             end
69         end
70     loop
71 end

```

Figure 60: Vehicle movement in pseudo code

Disabled roads

The City program allows the user to disable roads. This way a roadblock can be simulated. Roadblocks can occur in case of a traffic accident or roadworks, for example. To disable a road, click on it with the right mouse button. A message box will popup (for every road under the mouse pointer) asking whether to disable the road or not. Disabled roads are coloured red instead of black. The Routing system will also be notified of any disabled links, so it can compute alternative routes. Disabling roads will of course affect the vehicles in the traffic simulation. First of all the vehicles that are on a road when it is disabled are stopped. They will not be moved until the road is enabled again. Secondly there are vehicles that would like to take a route via a disabled road. How this is dealt with is explained with the aid of Figure 61. This flow diagram is actually an extension of the 'Vehicle.Move' procedure in the former paragraph.

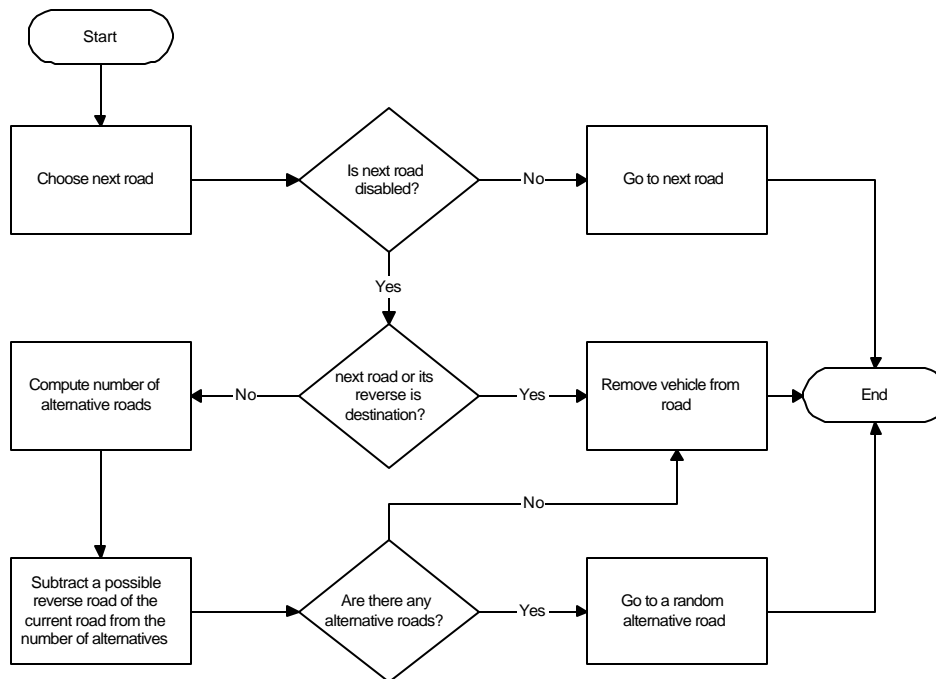


Figure 61: Flow diagram of vehicles dealing with disabled roads

In easy words the flow diagram explains that a vehicle about to take a disabled road will choose a random alternative road. If the vehicle has its destination at the disabled link, the vehicle is removed from the network, assuming that the driver parks his car and travels the remaining distance by feet. If there are no alternatives links for the disabled link, but the disabled link was no destination for the driver, then the vehicle is also removed from the network. Although this might not be very realistic, it is a simple way to avoid the network from getting silted up. Going back from where the vehicle came from is not considered an alternative, because when the vehicle will arrive at the previous intersection it will be sent back to the point where the road was blocked again. This would cause the vehicle to get in a loop. Choosing a random alternative does not guarantee that the vehicle will not end up in a circling around but it is a better option. Vehicles that use the Routing system to determine their route will in most cases not just take a random alternative. They will not receive a route from the Routing system with disabled links on the way, because the Routing system is aware of the disabled links and provides an alternative. Because the Routing system uses probabilities to determine the best roads, it simply chooses the road with the second best probability when the highest probability belongs to a disabled road.

Vehicle communication

Some vehicles communicate with the Routing system. There are two purposes for communicating with the Routing system. The most obvious reason is of course to request a route. Some of the vehicles use a default route but others request a route from the Routing system. The route, that the Routing system returns, depends on the traffic load on the network. This way, heavily loaded roads can be avoided. Because the traffic load on the network changes over time, also the best routes change over time. Therefore the vehicles need to request a new route every now and then. When a vehicle receives a new route, the old route is overwritten and from then on the vehicle will follow the new route. The other reason for communicating with the Routing system is to provide the Routing system with new information about the load of the traffic network. Not all the vehicles provide new information and the ones that do, do not necessarily also request routes from the Routing system. The information that they send consists of the covered route, road per road, and the time it took them to cover that path. The Routing system uses this information to determine the load of the traffic network per road.

2. The computation of travel times by the Timetable updating system

After the vehicles have been moved, the Timetable updating system is instructed to compute the travel times for all links (again). New measurements have been received from the vehicles. These measurements are used to compute a new travel time for every link. The measurements are not directly used as the new travel time, but a weight function is used to combine several measurements, taking into account the moment of the measurement. This function is described in more detail in section 3.4.1. Also when no new measurements are available, the travel times still need to be computed again as the weight of measurements decreases in time.

3. The search for new routes by the Route finding system

When the Timetable updating system has computed new values for the travel times of the links, the Route finding system is instructed to execute a step in the Ant Based Control algorithm. Such a step consists of generating new ants, and after that moving all existing ants. These ants can be forward ants as well as backward ants. The forward ants move from their source searching to their destination. When arrived at their destination the forward ants are transformed in backward ants. The backward ants follow the same route as the respective forward ants, but in the opposite direction from destination to source. See section 3.4.2 for a detailed description of this algorithm.

Communication messages

Below you will find a description of the message accepted by the Routing system. The messages are grouped by their function. The first group takes care of setting the parameters to the desired values. These values are added to the messages.

- set max seconds
- set max runs
- set ant steps per second
- set constant a
- set constant b
- set exploration probability
- set travel time differences
- set update subpaths

The next messages are generated by the vehicles in the traffic network. One message provides the Routing system with new information about the load in the network. The other message is used to obtain a route from one location to another.

- update routing system
- request route

The following two messages are used to notify the Routing system of changes in the traffic network.

- disable link
- enable link

The next messages are used to run and stop a simulation.

- do timestep
- reset

The following messages are used for mutual communication between distributed components. Here 'ds' stands for a tenth of a second, the accuracy of a measurement. With the ants also their memory is sent in the message.

- add forward ant
- add backward ant
- add measurement_ds

The last group of messages are used for statistical purposes, such as the construction of graphs.

- get number of ants
- get number of backward ants
- get total ant age
- get number of differing next nodes


Simulation speed control

To run the simulation, use the buttons on the upper right part of the City program (see Figure 62). Before the start of a simulation the stop button will be down. The second button from the left will execute one step of the simulation and then switch to the pause state (fifth button). The play button (third button) will run the simulation at normal speed. This speed can be adjusted with the scrollbar on the right of the buttons. The fast forward button (fourth button) will run the simulation at maximum speed. The simulation will not update the visual network in this state. The pause button will pause the simulation.



Figure 62: The buttons that control the running of a simulation

4.4.3 Evaluating the simulation

During and after a simulation nine different graphs are available for evaluating the results of the simulation. They can be opened by pushing one of the -buttons in the City program. Moving the mouse cursor over the icons will show a hint about the indicated graph. The available graphs are:

- Total number of ants
- Total number of backward ants
- Average living ant age

- Total number of differing next nodes
- Average standard vehicle age
- Average smart vehicle age
- Total number of vehicles
- Average standard route time
- Average smart route time

See section C.2 in the appendices for further details about graphs.

4.5 Special functionality

Here we will discuss two simple structures used for storing data during program execution. These structures had to be added because they were missing in the standard libraries, at least in the right format. The first structure is an integer queue. It can be used to store integers and fetch them again, first in first out. The other structure is an integer ring. It can be used to store integers and fetch them again in any order. But when data is added to a full ring, the oldest values are automatically overwritten.

4.5.1 Integer queue

Internally an integer queue is an array of integers with a fixed length, assigned at creation. Additionally there is a pointer to the first element of the queue (not necessarily the first position of the array), a size of the current queue and a maximum size of the queue. Figure 63 is a representation of an integer queue at a random time. The maximum size of the queue is 10, the current size is 7, and the first and oldest element is 24.

24	3	19	7	41	12	19			
----	---	----	---	----	----	----	--	--	--

Figure 63: An example of an integer queue

The allowed operations for the queue are:

- *Push*
Add an integer value to the end of the queue, after 19 in the first empty space. Adding to a full queue will raise an error.
- *Pop*
Fetch the first and oldest value from the queue and remove it from the queue. In this case 24 is returned and removed from the queue. Now 3 is the first and oldest value and an extra space becomes available.
- *Clear*
Remove all values from the queue. Now all 10 spaces are free again.
- *Read index*
Return the value at any given position. For example, reading index 4 returns 7. The value is not removed from the queue.

4.5.2 Integer ring

Internally an integer ring is an array of integers with a fixed length, assigned at creation. Additionally there is a pointer to the first element of the ring (not necessarily the first position of the array), a size of the current ring and a maximum size of the ring. Figure 64 is a representation of an integer ring at a random time. The maximum size of the ring is 8, the current size is 5, and the first and oldest element is 24.

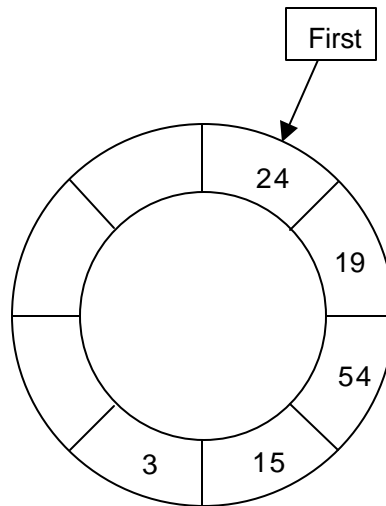


Figure 64: An example of an integer ring

The allowed operations for the ring are:

- *Push*
Add an integer value to the end of the ring, after 3 in the first empty space. Adding to a full ring will overwrite the first and oldest value, in this case 24. Afterwards 19 will be the first value.
- *Pop*
Fetch the first and oldest value from the ring and remove it from the ring. In this case 24 is returned and removed from the ring. Now 19 is the first and oldest value and an extra space becomes available.
- *Clear*
Remove all values from the ring. Now all 8 spaces are free again.
- *Read index*
Return the value at any given position, counted from the first. For example, reading index 4 returns 15. The value is not removed from the ring.

Chapter 5: Exploiting the applications

This chapter should give an impression of the possibilities and the limitations of the applications we developed: the City program and the Routing system. The City program is a simulator of a traffic environment in a city. We will explain what aspects from a real traffic environment can be realised in the simulator and what restrictions there are. For a description how to do this we refer to the user manual in appendix C. The Routing system is a guide for drivers that navigates them through a city. How well it works for this purpose will be discussed in the next chapter, but here we will go into some of the performance issues considering the computational power of one or more PC-systems.

5.1 City program

First we will discuss the possibilities of the City program. And afterwards we will point out some limitations of the program.

5.1.1 Possibilities

We will discuss the possibilities to create a traffic environment from a real city using Beverwijk as an example. Figure 65 shows a picture of Beverwijk from bird's-eye view [www.prentenkabinetbeverwijk.nl].



Figure 65: Picture of Beverwijk from bird's-eye view

Map creation

Of course this picture is not the easiest starting point for creating a virtual traffic environment. A better way is to use a 2D-map of the city. This map should at least contain the roads that we want to realise

in the virtual traffic environment. Figure 66 shows the map used to create an environment for Beverwijk. From this map the intersections can be measured in x - and y -coordinates. Then the roads can be added as connections between two intersections. The roads will be displayed as a straight line between the two intersections that it connects. Extra intersections placed in the curves could be used to approximate the curvature of a road. This would improve a realistic view, but it is not necessary for the accuracy of the simulation. And many extra intersections may cause the simulation to work slowly. More important is the construction of one- and two-way roads. Most roads are bi-directional, which means there is traffic possible in two directions. In that case two separate roads have to be created in the traffic environment, one for each direction. This obviously allows for the creation of one-way roads. Another topic that directly associates with this is the number of lanes per road. This information is often not available from many maps. This is however important information for the virtual traffic environment, because it significantly affects the capacity of a road. The number of lanes can be set per road. If the number of lanes changes halfway of a road, an extra intersection could be introduced that splits the road in two. A vehicle will automatically use the second lane when the first lane is occupied. Another matter that influences the capacity of a road is the length. To limit the maximum number of vehicles on a road, it is divided in pieces with a fixed length. There cannot be more than one vehicle per piece of a road on each lane. The length of a road also affects the travel time of the vehicles on that road, obviously. The road lengths can be set manually, but they can also be computed from the given intersection coordinates using the Pythagorean proposition. The obtained lengths can be multiplied by a constant factor. This will give reasonable results for most roads, but some manual adjustment will be necessary for curved roads. The traffic environment that was created from the 2D-map in Figure 66 is shown in Figure 67.



Figure 66: A map of Beverwijk

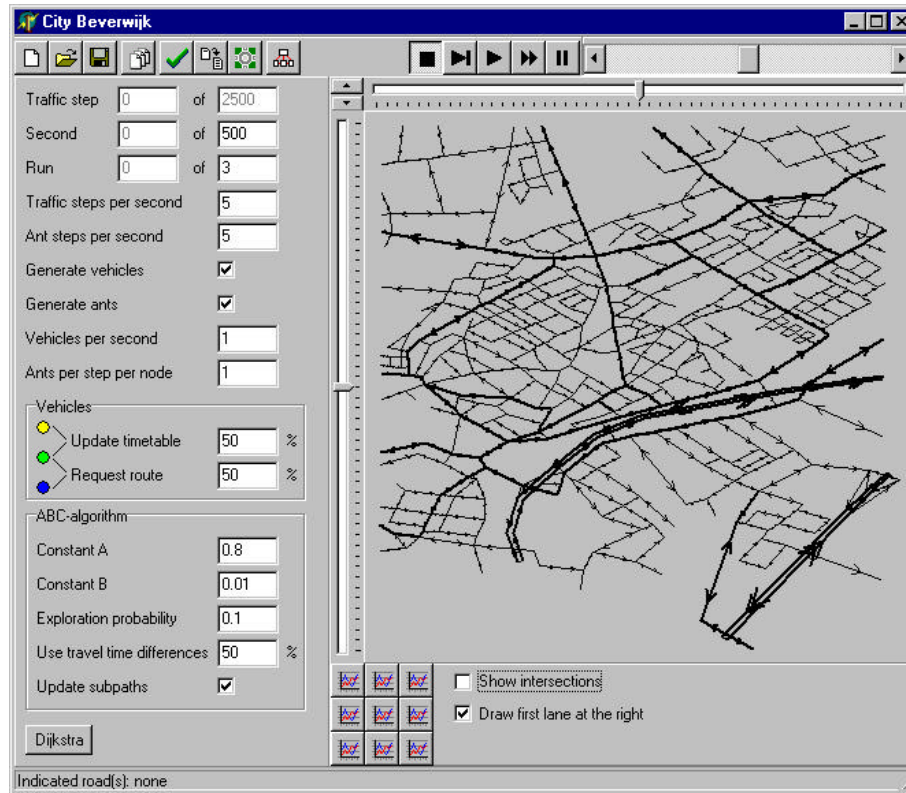


Figure 67: Virtual traffic environment of Beverwijk

Traffic rules

For intersections we distinguish three types: normal, traffic lights and roundabouts. At normal intersections just the usual precedence rules apply. Traffic on a road with priority gets precedence over traffic on a road without priority. And traffic from the right precedes other traffic from roads with the same priority. This implies the ability to assign a priority (high or low) to all roads. Intersections can also be controlled with traffic lights. Only fixed time control is possible, but for any road approaching the intersection the amount of time that a traffic light is green can be set per direction (left, right, ahead). For convenience a reasonable default is given. When the traffic lights for traffic from different directions are green at the same time, normal precedence rules apply again. The last intersection type is the roundabout. Roundabouts in the traffic environment will be transformed to a set of extra roads and intersections that are connected counter clockwise. On these new roads and intersections the normal precedence rules for roundabouts are applied to the traffic.

Traffic flow

The speed at which the vehicles in the traffic environment will drive is determined by the given speed per road. Data from local authorities can be used to compute realistic values for the speed per road. Available data can be some speed measurements of the vehicles at some point of the road. Another approach is to use a map with the maximum allowed speed per road. The most important aspect of the traffic flow might be: where it flows, i.e. where do the vehicles start, where do they go and what route do they follow. Where the vehicles start and where they go, can be set by assigning a rate to each road. The higher this rate the more vehicles will start at a road. The same goes for the destination of the vehicles. The route they follow is given by the Routing system or read from default routing tables. These default routing tables can be changed manually or computed by using Dijkstra's algorithm for shortest paths. Because vehicles get a random starting point and a random destination according to the given rates, it cannot be determined on forehand what roads will be used. So it is not possible to assign a road, for example, 50 vehicles per hour. A different approach is required. When an experiment

should simulate an afternoon rush hour, a high rate for starting vehicles should be assigned to roads in a region with a lot of companies. And when an experiment should simulate a situation on a time that many people go out shopping, the destination rates should be high at places with car parks in the city centre. The rates should also be high for highways, that is, only the point where they enter and leave the borders of the traffic environment. The total number of vehicles in a simulation will be zero at the start. And then a parameter determines the number of vehicles added per second. So in the beginning the total number of vehicles will only grow. And after some time when the first vehicles have reached their destination the number will be stable on average. Some testing is needed to adjust the parameter to achieve the desired load on the traffic network. Figure 68 shows the traffic in the simulation some time after the start of an experiment. The different coloured circles represent vehicles with different communication with the Routing system.

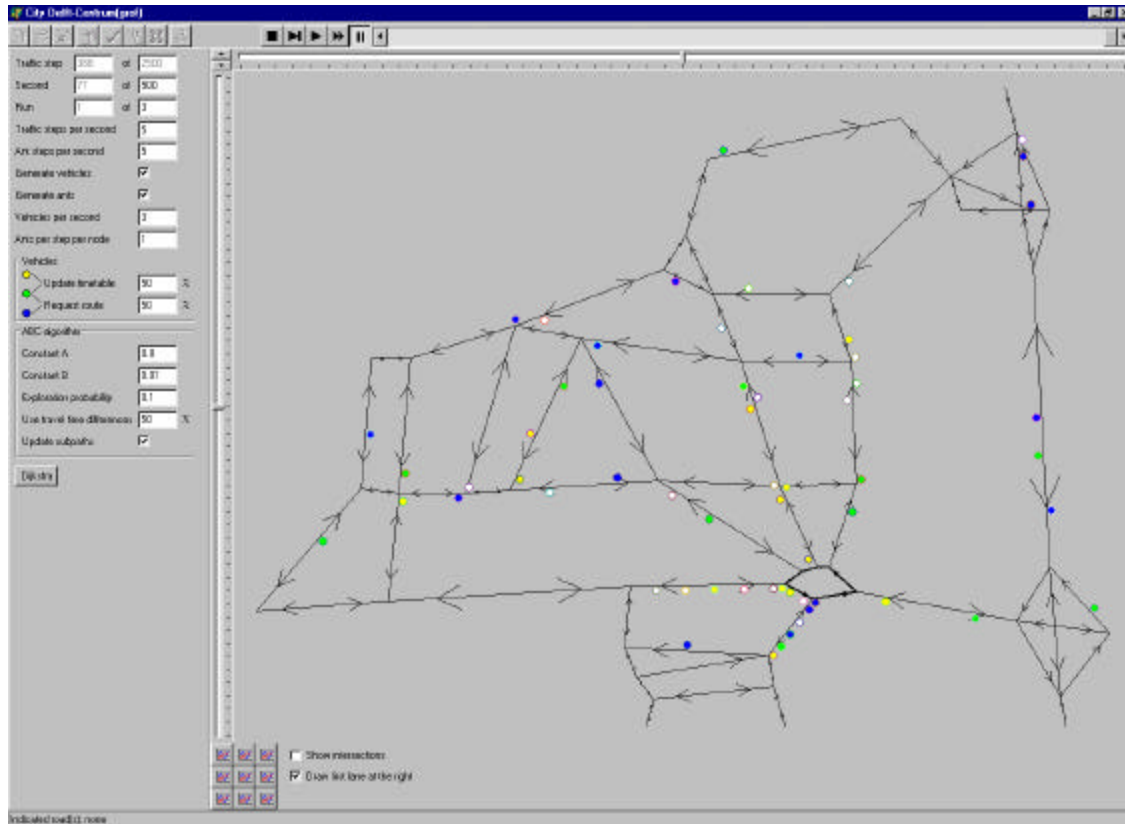


Figure 68: The traffic in a simulation after some time

5.1.2 Limitations

The simulation of the traffic is not perfect. Some aspects are simplified as opposed to real life traffic for several reasons. Too much detail would lead to slow simulation. Another reason for the constraints is the fact that the simulation is not the main goal of this thesis. The emphasis should be on the ABC-routing. This means that some choices are made for the sake of convenience. But there is also the proof of concept: using too much rules, variables and parameters may lead to the risk of not being able to spot the problem when things go wrong. Below a list is given of the most obvious constraints.

- *All vehicles are the same:*
All vehicles move with the same speed, acceleration and deceleration. There is no difference in the type of vehicles or the driving behaviour of the drivers.
- *Congestion only appears in front of intersections:*

A traffic jam will always start with a car waiting in front of an intersection. Following cars will be waiting for the predecessor vehicles. A curve in the road will not be a reason for delay. Just as other traffic that breaks to park or that leaves the road will not be a reason for any deceleration.

- *There are no other traffic participants than the standard vehicles:*
Cyclists, pedestrians and other possible traffic participants are not modelled. Traffic lights do not turn green for them. Nor does any driver have to stop for pedestrians on a zebra crossing.
- *Distances to the predecessor are independent of speed:*
A vehicle will keep the distance to his predecessor equally large regardless of his speed and the speed of the other car. When the driver in front has to stop, he will stand still immediately, just as the following car will stop directly in response to the first car.
- *When a deadlock situation occurs the first vehicle in a time step is allowed to drive:*
In case all vehicles waiting in front of an intersection are not allowed to drive according to the precedence rules, a deadlock would occur. When such a situation is detected, the first of those vehicles that will do a time step is given permission to go ahead. In a real-life situation the boldest driver would probably go first.
- *Traffic lights use a fixed time control:*
The green time of the traffic lights is not influenced by the traffic that is or is not approaching the light. There is a fixed time that the lights are green and red. There is also no yellow light state for the traffic light.
- *A vehicle can choose a direction independent of its current lane:*
When a vehicle drives on a road with multiple lanes, it does not matter which lane it chooses. Even when a driver chooses the left lane on a three-lane road, he can turn right without waiting for vehicles on the other two lanes.
- *Turning back can only be done at intersections:*
Guided drivers who receive a route, which tells them to turn around and go back, and unguided drivers that notice a congestion, might want to turn half way between two intersections. In real life this would sometimes be possible, sometimes turning is impossible even at the end of a link. The simulation allows for turning at the end of a link, assuming the road is bi-directional.

5.2 Routing system

The Routing system uses a map of a traffic network and recent route information of vehicles to compute fastest routes through a city. It can consist of one application or a set of applications working together by communication. First we will discuss this distributed character, then we will address some performance issues.

5.2.1 Distributed Routing system

The routing task of the Routing system can be distributed over any number of applications in the range from 1 to the number of intersections in the traffic network. This means that the task does not have to be executed on one computer. For example the routing for a city can be split up in sectors. Each sector should be a natural subdivision such as a district of a city. Figure 69 and Figure 70 show the communication of the City program with different configurations of the Routing system. Every Routing system part takes care of some part of the traffic network. New information from vehicles about the state of the traffic network is sent to one of the Routing system parts. This information is processed and if necessary sent to the appropriate Routing system part(s). The ants, which take care of the computation of the shortest paths, move through the Routing system. So they are also sent to other parts of the Routing system if necessary. And when a vehicle requests a route, this request is sent to one of the Routing system parts. This part may not know the entire route for the vehicle. In that case the remaining part of the route is requested from other parts of the Routing system. Hence some communication is necessary between the different parts of the Routing system. This means that some consideration has to be taken for the optimum number of Routing system parts.

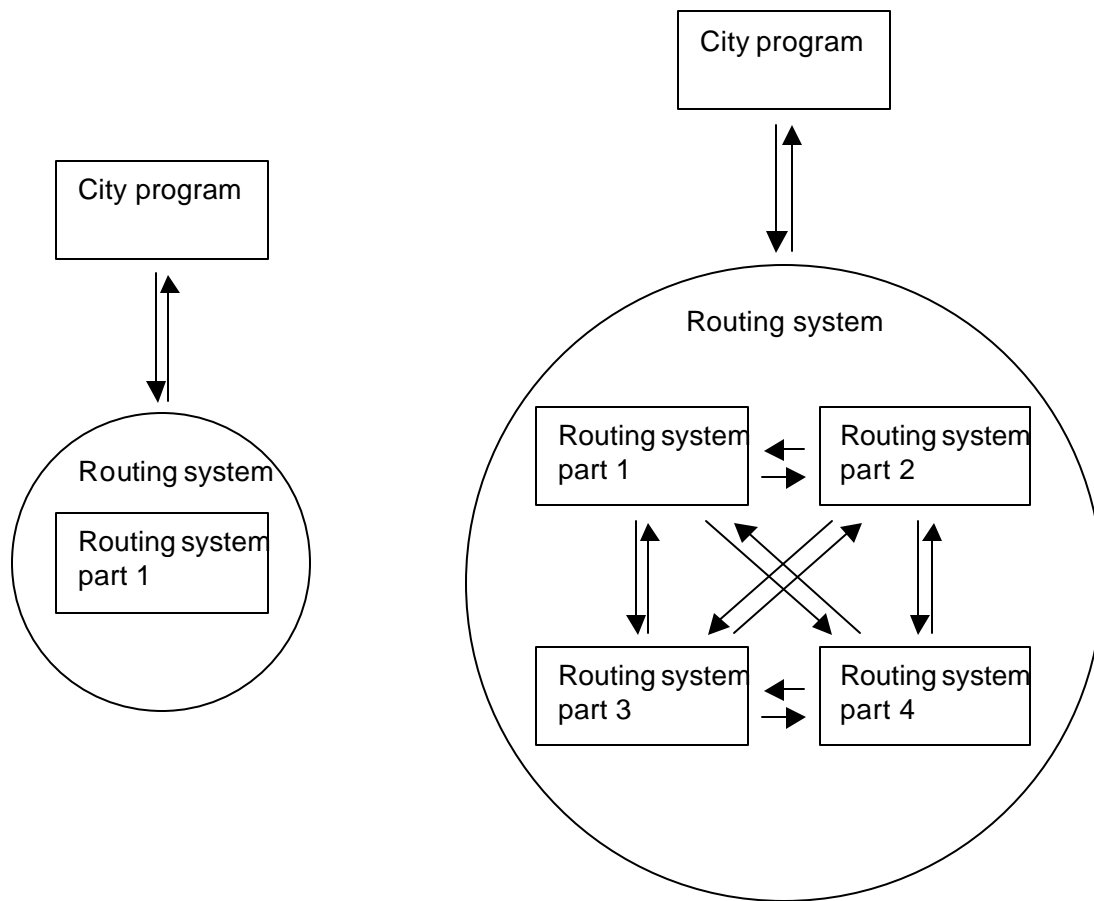


Figure 69: Routing system consisting of one part

Figure 70: Routing system consisting of four parts

TCP/IP is used for communication between the City program and the Routing system. For this communication an address is needed consisting of an IP-address and an IP-port. For the ease of setting up a Routing system all parts of the Routing system currently get the same IP-address as the computer where the City program is running. And accordingly all parts of the Routing system are (automatically) started at the same computer. Only the IP-port numbers are different. This means that both the City program and all parts of the Routing system are currently only able to work on the same computer. Therefore no benefit can be gained by running several Routing system parts. This allows us however to set up a Routing system by just pushing one button.

5.2.2 Performance issues

We will now discuss which environments can be simulated and which environments are no longer appropriate because of their size. Keep in mind that the Routing system is prepared to be distributed over several computer systems, but can only be executed on one computer at the moment. So the City program and the Routing system will use the same computer concurrently. Tests have indicated that the City program consumes about 20 % of the CPU-time and 80 % of the CPU-time is consumed by the Routing system when a computer system is loaded to its maximum. For these tests only one Routing system part is used, because more parts only give an unnecessarily extra load on one computer. The most likely reason for the Routing system to use more CPU-time than the City program is because there are more ants in the Routing system than vehicles in the City program. The City program does not only provide the possibility to change the speed of the simulation, it also provides

the option to run at maximum speed without visual changes. This feature is added in case that screen updating might be too slow. However, the gain in the maximum speed of the simulation is less than 1 % when not updating the screen. The minimum requirement for the simulation speed is about real-time speed. This means that simulating traffic for one minute will take one minute on the computer. More desirable would be that a simulation could be speeded up, so that we can run an experiment of, for example, an hour in only ten minutes. In that case the speedup-factor would be 6.

It may be clear that a larger network implicates a lower maximum simulation speed. This is true for several reasons. The most important reasons are related to the Routing system, because it will consume most of the CPU-time. First of all, because ants in the Routing system will be generated per node, more nodes imply more ants in the Routing system. Secondly a larger network leads to larger distances to be covered by the ants. And the longer the distances, the older the ants get. And the older the ants get, the more ants will reside in the Routing system concurrently. A larger network also provides more possible alternative routes for the ants. And so the ants have even longer routes to cover. This also contributes to a larger sum of present ants. It will be clear that more ants consume more CPU-time.

A good indication of the size of the network is the number of nodes. We have tested the speed on a computer with a 1000 MHz CPU and 256 MB RAM. An experiment with a traffic network of 50 nodes resulted in a maximum speedup-factor of 2.7. This is an acceptable value for experimenting. A more realistic network of a small city containing 500 nodes provided a speedup-factor of 0.4. This means that simulating one minute of traffic will take 2.5 minutes. Such a network size is not suited for experimenting on this computer.

Chapter 6: Experiments and results

In this chapter we will discuss the results of some experiments. The experiments were executed in different environments and with different parameter settings. The results tell us something about the quality of the simulation and, more important, about the quality and effectiveness of the Routing system. The used environments are a little small to be very realistic and they might appear a little artificial. This is done for the speed of the simulations, to be able to execute many runs. And these environments are easier to check and to predict. Another reason is that large realistic environments take a lot of time and data before some realism is accomplished. For all experiments we will describe the goal, the environment with the settings, the expectations and the results.

6.1 Experiment 1: Adaptation speed

6.1.1 Goal

With this experiment we want to test the adaptability of the Routing system when some roads are disabled. When the Routing system is in an optimal state and the state of the traffic network changes, the Routing system has to adapt to the new situation. This is done by the ants: they constantly move through a virtual traffic environment and change the probability tables. By judging whether they have found good routes, they increase the probabilities for that route more or less. We would like to measure the time it takes for the Routing system to adapt to the new situation.

6.1.2 Environment and settings

The traffic network is a grid with 4 x 4 intersections (see Figure 71). All the vehicles in the simulation will drive from the road between intersections 1 and 2 to the road between intersections 15 and 16 or vice versa from 15/16 to 1/2. This is accomplished by setting the source and destination rates of all other roads to zero. Because all roads have the same length and maximum speed there are four possible routes the traffic could take to accomplish the shortest travel time. Indicated by the numbers of the passed intersections, these routes are:

- 2 – 3 – 7 – 11 – 15
- 2 – 6 – 7 – 11 – 15
- 2 – 6 – 10 – 11 – 15
- 2 – 6 – 10 – 14 – 15

The vehicles driving from the road between intersection 15 and 16 to the road between intersections 1 and 2 should choose the reverse order of one of these routes. When computing the shortest path with the built-in Dijkstra's algorithm, the first one is chosen as the shortest route (in time). This does not mean that the other routes are longer, but just one has to be chosen and this happens to be the first. This route will be used by all vehicles that do not use the Routing system. The vehicles that do use the Routing system will initially also use this route, because the initial state of the Routing system is copied from these static routes. So in the beginning all vehicles will be driving via intersections 2, 3, 7, 11 and 15 (or in reverse order). Now we will disable the roads between intersections 7 and 11 (one road in each direction) as if there was a roadblock because of an accident or roadworks. The vehicles that do not use the Routing system will choose a random alternative when arriving at the blocked road. In most cases this means that the vehicles will take a longer path than necessary. The vehicles that do request the Routing system for a route, will probably make that same mistake at first. But the Routing system can adjust this route dynamically. We will initially use the default parameters for this simulation. These defaults are:

- Traffic steps per second = 5
- Ant steps per second = 5
- Generate vehicles = True

- Generate ants = True
- Vehicles per second = 1.0
- Ants per step per node = 1.0
- Update timetable = 50 %
- Request route = 50 %
- Constant A = 0.8
- Constant B = 0.01
- Exploration probability = 0.1
- Travel time differences = 50 %
- Update subpaths = True

We will also change these parameters, one at a time, to find out what the most appropriate settings are. For every set of parameters we will do ten runs of the simulation and then compute the average. What we will measure is the time (in seconds) when the first vehicle, which is on its way from intersection 1/2 to intersection 15/16, will chose intersection 6 instead of intersection 3. When a vehicle drives via intersection 3 it will have to make a longer route than necessary, because the road between intersections 7 and 11 is blocked. When a vehicle drives via intersection 6 instead, it can avoid the blocked roads without making a longer route (see Figure 71).

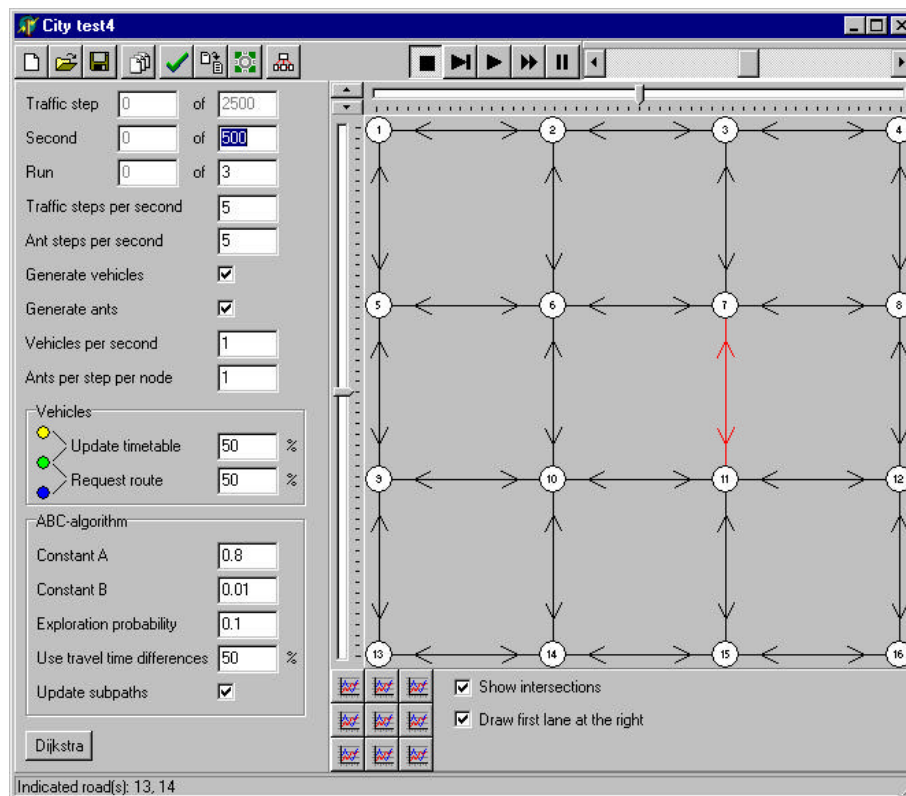


Figure 71: Traffic environment for experiment 1

6.1.3 Expectations

From the ABC-algorithm we know that to find an alternative route many ants have to follow that route and deposit their pheromone along this route. Initially the probability to chose intersection 3 from intersection 2 will be high, because this is copied from the static routing algorithm of Dijkstra. But when the ants follow this route their travel time will higher than when following an alternative route via intersection 6. In the beginning most ants will chose the longer route, because of this high probability. But there will also be some ants that chose a shorter route. The ants with a shorter route

will increase the appropriate probability more than the ants with a longer route. At some moment the probability for taking the shorter route will become higher than the probability for the longer route. From that moment vehicles that request a route from the Routing system will be guided along the shorter route via intersection 6 instead of intersection 3, because the vehicles follow the route with the highest probability. When a vehicle receives this new route and then reaches intersection 2 it will chose intersection 6 as the next intersection to go to. The first occurrence of this event will be measured.

6.1.4 Results

The results of the experiments with this environment are enumerated in Table 6. In the first series of ten runs we left all parameters to the default values. In the rest of the series we changed only one of the parameters at a time. This allows us to compare the results of the other series with the results of the first. The results of the separate runs are given in appendix B.1.

Table 6: Results of experiment 1

Series	Description	Average	Standard deviation (with n-1)
1	default parameters	192.8 s	83.4 s
2	no updating of sub paths	422.2 s	303.0 s
3	2 ants per step per node	121.8 s	60.4 s
4	10 ant steps per second	99.6 s	87.1 s
5	constant A = 0.6	184.9 s	82.1 s
6	constant A = 1.2	157.9 s	111.2 s
7	constant A = 1.6	126.7 s	56.4 s
8	constant A = 2.0	129.0 s	79.0 s
9	constant A = 3.0	77.7 s	48.8 s
10	constant A = 4.0	71.7 s	51.7 s
11	constant A = 5.0	63.3 s	25.1 s
12	constant A = 6.0	41.9 s	15.2 s
13	constant A = 7.0	59.5 s	34.6 s
14	constant A = 8.0	61.1 s	54.5 s
15	constant A = 9.0	70.7 s	55.8 s

Default parameters

When all parameters are left to the default, the first vehicle that takes the shorter route is noticed after a little more than 3 minutes (on average). This is a rather good result but it can be enhanced by adjusting the parameters. The standard deviation shows that the result may vary quite a bit. There are two main factors that cause this. The first is the randomness of the ants: they walk through the virtual traffic network using the probabilities to determine their path. And the second factor is the randomness of the vehicles: when there is no vehicle, neither will there be a vehicle to take the short path. This may occur when there are more vehicles coming from intersections 15/16 than from 1/2 (50 % on average). Another reason can be that the vehicles coming from intersections 1/2 do not use the Routing system to be guided (50 % on average). And finally a traffic jam can stop vehicles from arriving at intersection 2.

No updating of sub paths

Figure 72 shows a graph of the average living ant age. This graph is the result of three runs of 500 seconds of the simulation with default parameters. The average living ant age is a little more than 3 steps. This means that the average ant is killed after 6 to 7 steps (assuming a uniform distribution of

the maximum ant age). The length of the path of a forward ant is half the path of the forward ant and backward ant together. So the average forward ant makes a path of a little more than 3 steps and then returns as a backward ant. The backward ant updates all probabilities to the destination of the forward ant at the nodes that it passes. So this would be a little more than 3 updates on average for this case. Now when the backward ant also updates the sub paths, he would update 6 probabilities instead of 3 (see Figure 73). This improvement is very noticeable, because leaving this option out results in poor behaviour of the Routing system in this experiment. It takes more than twice as much time to adjust the Routing system to the new situation with the disabled roads.

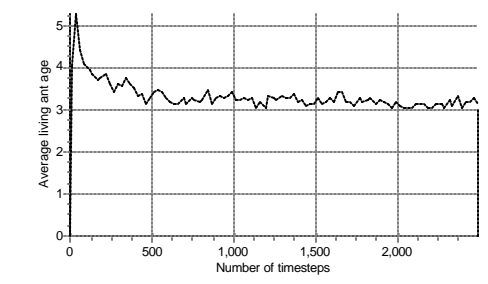


Figure 72: Graph of the average age of the living ants

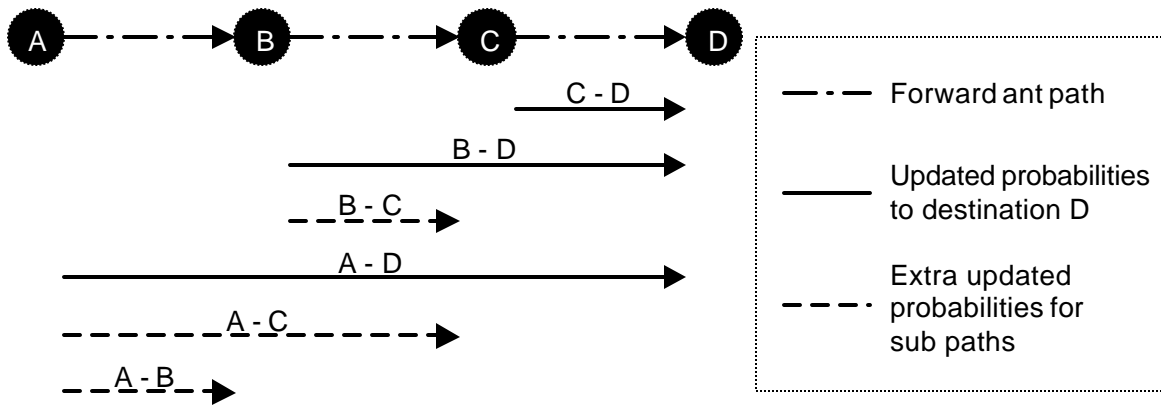


Figure 73: Updating probabilities for sub paths or not

For this experiment where the backward ants make three steps to get back (on average), twice as much probabilities are updated when also updating sub paths. But when the path of the ant is longer the difference is even greater. Table 7 shows the number of updates for some path lengths, when also updating sub paths.

Table 7: Number of updates with a given path length

Path length	Number of updates
1	1
2	3
3	6
4	10
5	15
6	21

For a path of length n the number of updates is n when no sub paths are updated. But when also updating sub paths the number of updates can be computed as follows:

$$t_{n+1} = t_n + n + 1,$$

$$\text{with } t_1 = 1$$

Using the theory of difference equations, this can be rewritten to the following function:

$$f(n) = \frac{1}{2}n^2 + \frac{1}{2}n$$

So the effect increases quadratic with longer paths. Note that the advantage affects the shorter paths more than the longer paths. In our experiment the longest optimal path has a length of 6 steps (from intersection 1 to intersection 16 for example). So the path to be adjusted (from intersection 2 to intersection 15) is relatively long (4 steps), but even then this option is an important improvement.

2 ants per steps per node

Generating 2 ants per step per node instead of 1, results in a considerable improvement. Using 1 ant per step per node caused the Routing system to be adjusted after more than 3 minutes. And the Routing system needed only about 2 minutes when using 2 ants per step per node. It can be expected that using even more ants would result in even better times. But there is a price for using more ants. More ants will use more CPU time and more memory and more communication is needed between the different parts of the Routing system. Because time and memory are sparse, the number of ants that can be used is limited. This limit depends on the size of the traffic network and the speed and memory of the computer system(s).

10 ant steps per second

Increasing the speed of the ants from 5 to 10 steps per second produces even better improvements than doubling the number of ants generated per second. This is because the ants do not only finish their route quicker but there are also generated twice as much ants per second. When the number of ants generated per step (per node) is the same and there are more steps (per second), then there are more ants generated too. This explains why the results are even better than when doubling the number of ants to be generated. Of course this improvement has the same price for time and memory as increasing the number of ants.

Changing constant A

The choice of 0.8 for constant A as the default value was rather random. Lowering this constant to 0.6 appears to give some improvement but it is expected to be caused by randomness and the rather small sample size of 10 runs. Increasing the value resulted in more spectacular improvements. A value of 6.0 gave the best results. In our traffic environment we are interested in the change of the probability for destination node 15 in the probability table of node 2 (Figure 71). The ants will register a minimal path of 4 steps to get there. The minimum travel time per step/road is 23 seconds according to the given length and maximum speed. So the virtual travel time of the ants from node 2 to node 15 is about 92 seconds, assuming that the vehicles do not suffer from too much delay. For the computation of the update of the probability not only the travel time is used, but also the extra time caused by a delay. This delay has shown to be irrelevant on the path from node 2 to node 15. Because the setting 'use travel time difference' is set to 50 %, the time that is used for the computation of the update of the probability is about 46 seconds (50 % of 92 + 50 % of 0). This time is used in the computation of Δp in the formula:

$$\Delta p = \frac{A}{t} + B$$

The best results were achieved when constant A was 6.0. The constant B was left as the default value of 0.01. This results in a value for Δp of 0.14. So a backward ant updates the probability at node 2 for destination 15 with steps of approximately 14 %. This is a rather high value that results in quick adjustments. But there is also a risk of instability, which causes the probabilities to change according to the randomness of the ants. The choice of the value for constant A is very dependant of the environment. Higher values of A are better suited for large paths and lower values are more appropriate for shorter paths. Herein lies an opportunity to enhance the used algorithm to update the probabilities.

6.2 Experiment 2: Effectiveness

6.2.1 Goal

The goal of this experiment is to determine the effectiveness of the Routing system. In other words the question is: How much travel time reduction can we gain by using the Routing system? The function of the Routing system is to navigate vehicles via the shortest roads in order to minimise their travel time. We assume that some vehicles use this Routing system and others do not. We assume that the vehicles that do not use it, follow fixed routes, which are not adapted to the dynamic situation of the traffic. Vehicles navigated by the Routing system should be able to avoid traffic jams and other causes of delay. So one way to compute the effectiveness is to compare the travel times of these two groups of vehicles. But when some vehicles are deducted from the heavily loaded roads this gives some relief to the vehicles on those heavily loaded roads too. So the vehicles that do not use the Routing system also profit from the Routing system. This leads to another way to compute the effectiveness: compare the average travel time of all vehicles without using the Routing system to the average travel time of all vehicles when some of them use the Routing system.

6.2.2 Environment and settings

The traffic network for this experiment is shown in Figure 74. The vehicles in this experiment have four possible starting points, namely the road between intersections 1 and 2, the road between 9 and 10, the road between 5 and 6, and the road between 6 and 7. There are only two possible destinations, namely the road between 1 and 2, and the road between 9 and 10. The start and end points are always at the middle of the road. Because all the start and end points are equally likely to be used by the vehicles, the following can be stated about the routes of the vehicles:

- 25 % of the vehicles will drive from 1/2 to 9/10 on average.
- 25 % of the vehicles will drive from 9/10 to 1/2 on average.
- 12½ % of the vehicles will drive from 5/6 to 1/2 on average.
- 12½ % of the vehicles will drive from 5/6 to 9/10 on average.
- 12½ % of the vehicles will drive from 6/7 to 1/2 on average.
- 12½ % of the vehicles will drive from 6/7 to 9/10 on average.

Intersection 6, which may be expected to be quite busy, is controlled by traffic lights. These traffic lights are used in the simulation to create extra congestion. The cycle time of the traffic lights is 60 seconds and the vehicles from all four directions each face a green light for 10 seconds. These green phases are separated with a period of 5 seconds, in which all lights are red. The roads between intersections 5, 6 and 7 have double lanes. We ran this experiment twice. The first time the parameters were set to the following values:

- Seconds = 2400
- Runs = 1
- Traffic steps per second = 5
- Ant steps per second = 1.0
- Generate vehicles = True
- Generate ants = True

- Vehicles per second = 1.0
- Ants per step per node = 1.0
- Update timetable = 50 %
- Request route = 50 %
- Constant A = 0.8
- Constant B = 0.01
- Exploration probability = 0.1
- Travel time differences = 50 %
- Update subpaths = True

This run allows us to compare the travel times of vehicles that use the Routing system to the travel times of the vehicles that do not use it. To compute the overall effectiveness of the Routing system on both types of vehicles a second run is executed. This time none of the vehicles will use the Routing system. The parameters are set as follows:

- Seconds = 2400
- Runs = 1
- Traffic steps per second = 5
- Ant steps per second = 5
- Generate vehicles = True
- Generate ants = True
- Vehicles per second = 1.0
- Ants per step per node = 1.0
- Update timetable = 50 %
- **Request route = 0 %**
- Constant A = 0.8
- Constant B = 0.01
- Exploration probability = 0.1
- Travel time differences = 50 %
- Update subpaths = True

Note that the use of ants and the parameters used by the Routing system are ineffective when there are no vehicles using the Routing system.

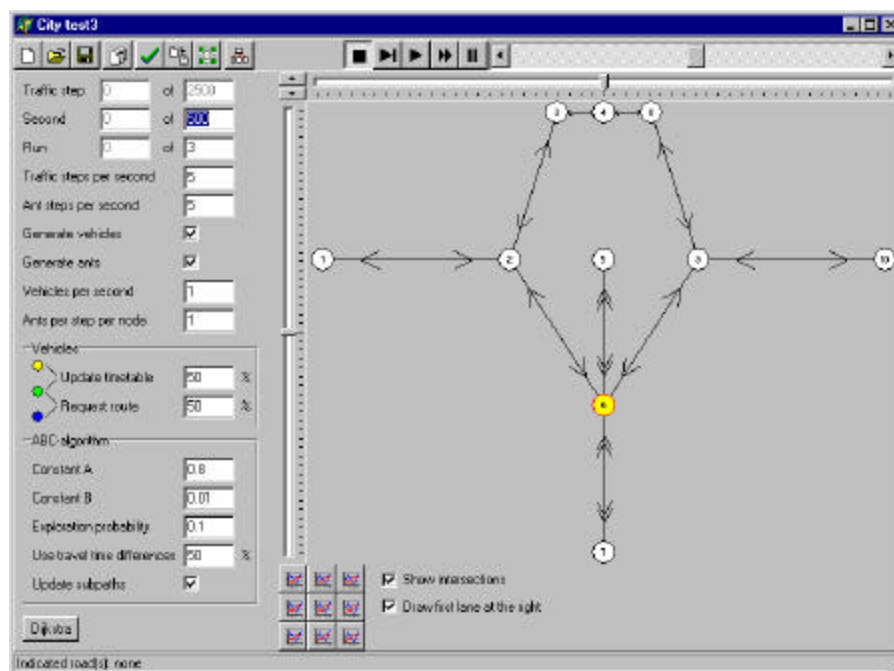


Figure 74: Traffic environment for experiment 2

6.2.3 Expectations

For vehicles from the road between intersections 5 and 6 or 6 and 7 there is only one reasonable path to their destination. The alternative paths, which go via intersections 3, 4 and 8, will hardly ever be more attractive for those vehicles. For the vehicles from the roads between intersection 1 and 2 and intersection 9 and 10 there are two reasonable options. They can take the northern path via intersections 3, 4 and 8, or they can take the southern path via intersection 6. Taking the length and maximum speed into account, the path for a vehicle from 1/2 to 9/10 can be covered in 75 seconds when passing intersections 2, 6 and 9. The alternative via intersections 2, 3, 4, 8 and 9 will cost at least 82 seconds. So all vehicles from 1/2 to 9/10 and vice versa will initially take the southern route. But intersection 6 of the southern route is a point where many vehicles from different roads join and cross each other's path. Therefore it is controlled by traffic lights. These traffic lights make the crossing safer and the priority for vehicles from different roads is distributed more fairly. On the other hand the traffic might perceive a considerable delay at this intersection because of the heavy load and the traffic lights. This will cause the northern path to be more attractive for vehicles from intersection 1/2 to 9/10 and vice versa. So we expect the vehicles that follow the advice of the Routing system to drive via the northern roads, where there are no delaying intersections. This should result in faster routes for these vehicles as opposed to the vehicles that do not use the Routing system.

6.2.4 Results

The first run of this experiment provided the following graphs (Figure 75 and Figure 76). These graphs show the differences in the average travel time of standard and smart vehicles over period of 40 minutes (2400 seconds). The standard vehicles do not use the Routing system, the smart vehicles do. The value at the end, after 2400 seconds, is the average of all measured travel times since the start of the experiment until the end.

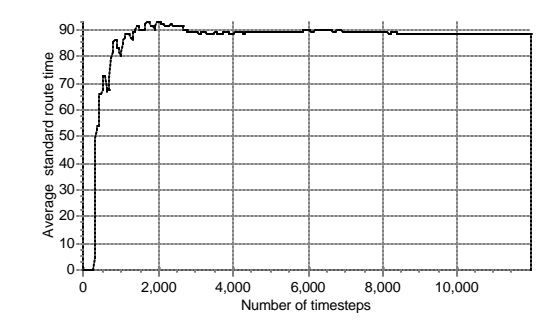


Figure 75: Average standard route time

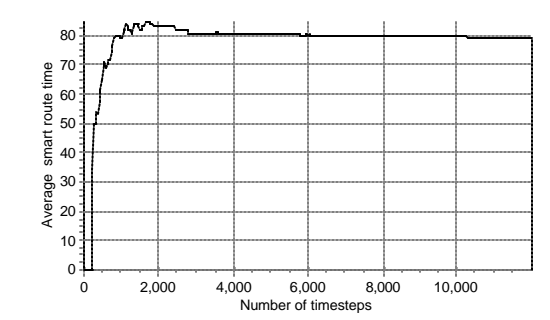


Figure 76: Average smart route time

When we zoom into the graphs (Figure 77 and Figure 78) we see that the (rounded) value for the standard vehicles is 88 seconds and the value for the smart vehicles is 79 seconds. So the overall profit for the smart vehicles is 10 % on average as opposed to the standard vehicles, which do not use the Routing system.

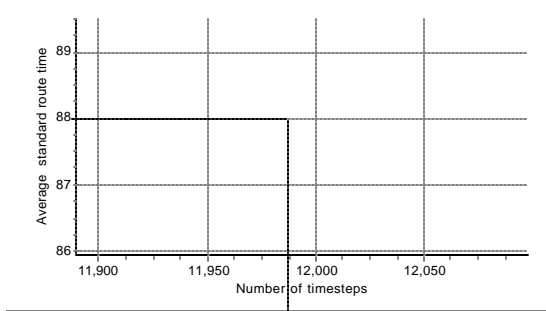


Figure 77: Average standard route time
(close-up)

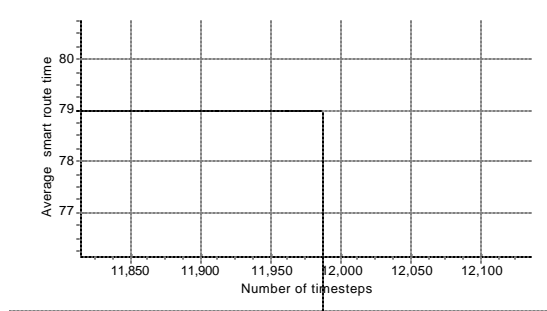


Figure 78: Average smart route time
(close-up)

Note that the differences are only caused by the vehicles driving from intersections 1/2 to 9/10 and vice versa, and these make up only 50 % of the smart vehicles. The other smart vehicles use the same paths as the standard vehicles. So the effectiveness for the smart vehicles driving from intersections 1/2 to 9/10 (and vice versa) against the standard vehicles with the same source and destination is even bigger. To be able to compare these paths, we have measured the travel time of all vehicles per road. The acquired average travel times (per road) are given in Table 24 of the appendix. With these averages per road we can compute the average path lengths for all paths of the vehicles. But we are interested in the differences between the northern path and the southern path. The results are shown in Table 8. Note that we computed the path from the middle of the road between intersections 1 and 2 to the middle of the road between intersections 9 and 10, because that was the complete path that the vehicles covered. The average profit for vehicles that used the northern path (because they were routed by the Routing system) is 19 % against the travel time of the standard vehicles.

Table 8: Travel time differences for the northern and southern path

	Northern path	Southern path	Profit for smart vehicles
From west to east	85.89 s	101.78 s	16 %
From east to west	83.05 s	105.51 s	21 %
Average	84.47 s	103.65 s	19 %

We have to bear in mind that these results are specific for this environment. We cannot say in general that using the Routing system provides this gain of time. We have created an environment in which considerable profits were to be expected. In real life most times these percentages are not realisable. Very often people that drive from home to work every day know the shortest routes by experience and they might not take very much advantage of the Routing system at all. For example, if you know there are traffic lights at intersection 6, you could avoid them without the aid of a Routing system. More advantage is possible in places that the driver does not know.

Another effect of the Routing system that we wanted to measure was the impact on vehicles that do not even use the Routing system. Therefore we ran the same experiment without the Routing system. Figure 79 shows the graph of the average route time. A close-up of the value at the end of the experiment shows that the average route time was 129 seconds. As we remember the average route time for the vehicles that did not use the Routing system was 88 seconds in the first run of the experiment. So in this period of 40 minutes the average profit for the standard vehicles is 32 %. The graph also shows us that the route times are still growing. This is explained with Figure 81, which is a screenshot of the City program at the end of the simulation. It can be seen that one of the roads to the intersection controlled with traffic lights (intersection 6, between 5 and 7) is completely saturated, causing even the vehicles on the road from intersection 1 to 2 to wait for the same red light. This congestion occurs because none of the vehicles take the alternative route via the north. Of course the

value of 32 % profit is specific for this environment and with this parameters, but it shows that a considerable advantage can be taken from the Routing system in some cases, even for vehicles that do not use the Routing system.

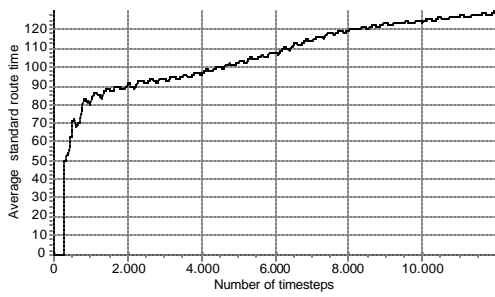


Figure 79: Average route time without Routing system

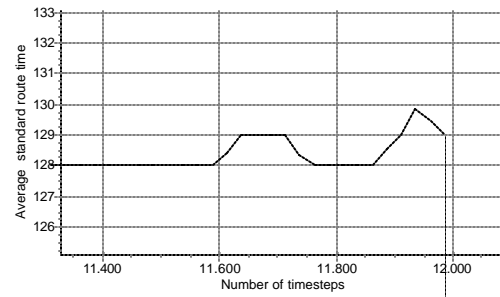


Figure 80: Average route time without Routing system (close-up)

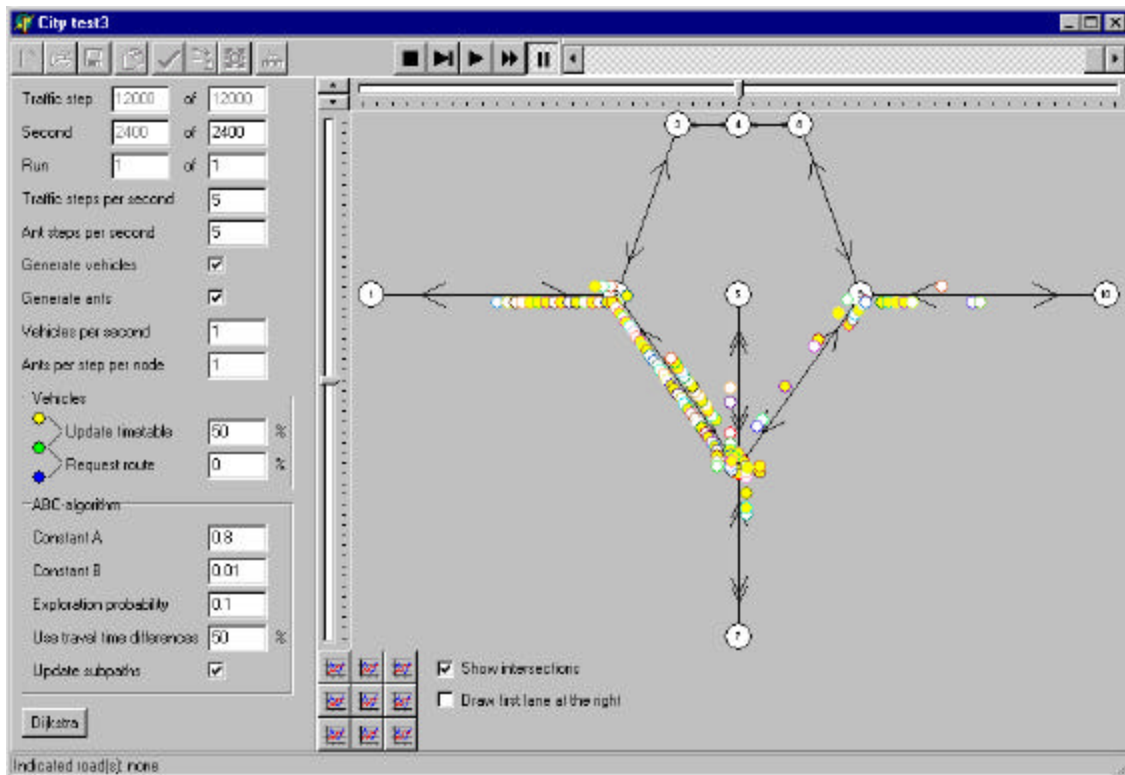


Figure 81: Screenshot of the City program after 2400 second without Routing system

6.3 Experiment 3: Stability

6.3.1 Goal

In experiment 3 we will examine the stability of the ABC-algorithm. The stability is the degree to which the system remains in the same state. In this case it means that the agents (or ants) in the Routing system continue to move and update probabilities, but the Routing system will provide the same routes for the same requests of the vehicles. The route that the Routing system returns depends on the probabilities in the probability tables of the nodes. The highest probability for a given destination will determine the next step of the route to that destination. So the system is stable as long as the highest probabilities remain the highest. We measure the stability of the system by computing the instability, i.e. how many times does the highest probability switch to an alternative node. Of course neither the stability nor the instability can directly be used to evaluate how well the Routing system performs. When the Routing system is completely stable, it does not adapt to changing network condition. So the vehicles will always obtain the same route, whether it is the shortest or not, whether it is congested or not. A high degree of instability does not guarantee good performance either, because this could mean that every vehicle receives a random route. What we want is a Routing system that quickly changes to the state where it provides the shortest routes in time and then stays stable until there really are shorter paths. So we will analyse the stability in combination with the speed of adaptation.

6.3.2 Environment and settings

First we will explain the environment used for our experiment and the applied settings. For our experiment we used the traffic network as shown in Figure 82. A similar network is often used to explain the principles of ant behaviour that we use as the base for our routing algorithm. According to that analogy the vehicles start at the road between intersections 1 and 2 (the nest) and go to the road between intersections 6 and 7 (the food). Because we only want to measure stability at one intersection, we are not interested in the vehicles that go back from intersections 6/7 (the food) to intersections 1/2 (the nest). Therefore we will not send vehicles in that direction. The vehicles from 1/2 to 6/7 really have two possible roads to choose: via intersections 2, 3, 5 and 6 or via intersections 2, 4 and 6. The same speed is allowed on all roads and all intersections are normal, i.e. the usual precedence rules apply and all roads have the same priority. So it is obvious that the southern road via intersections 2, 4 and 6 is faster (without congestion). The travel times for the two alternatives have a ratio of 55/45 with respect to the north/south route. As explained before we are interested in a stable Routing system that does quickly adapt to changes in the traffic network. To be able to evaluate this we have created an environment in which the Routing system has to adapt to a new situation, but the environment provides a reasonable alternative, so the stability is not obvious. For this purpose the vehicles are initially routed via the longer northern path. This means that the probabilities in the probability table of intersection 2 for destination 6 are set to:

- 0.6 for going to next intersection 3 (taking the northern route)
- 0.2 for going to next intersection 4 (taking the southern route)
- 0.2 for going to next intersection 1 (going back)

Because intersection 3 has the highest probability the vehicles will (initially) be routed along the northern path, which is not the fastest path. But the differences are not too big: 55/45. To visualise what the highest probability at some time is, we need vehicles that use the Routing system to be guided. The vehicles that do not use it are not interesting, so we set the parameter Request route to 100 %. We will not use vehicles that send update information to the Routing system, because congestion (caused by the application of the precedence rules) might disturb the relation of 55/45. So the parameter Update timetable is set to 0 %. The consequence is that all vehicles in the simulation will be coloured blue. The value of the parameter Travel time differences is set 0 %. Otherwise the ants not only collect travel times but also delays. In this experiment the delays are not measured, because no

vehicles send update information to the Routing system. Therefore it is useless to compute a delay. Changing this value does however affect the desired value for constant A.

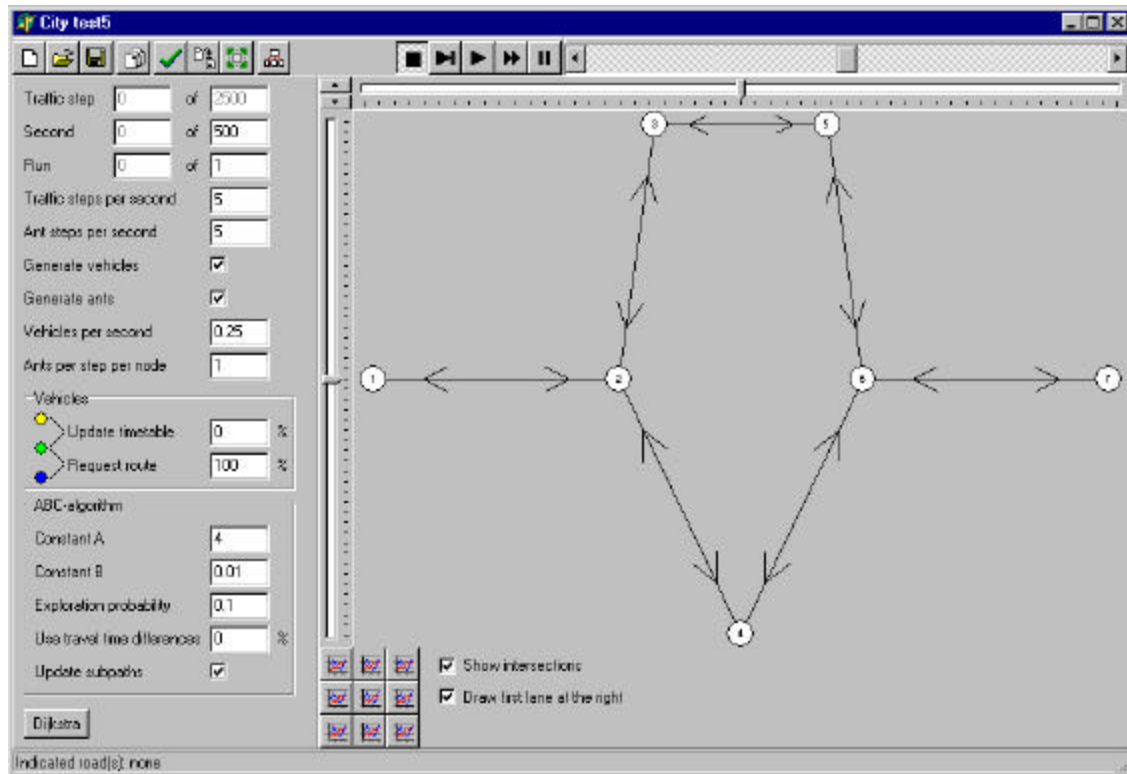


Figure 82: Traffic environment for experiment 3

We have run a number of different experiments, but the following parameter values have remained constant:

- Seconds = 120
- Runs = 1
- Traffic steps per second = 5
- Ant steps per second = 5
- Generate vehicles = True
- Generate ants = True
- Vehicles per second = 0.25
- Ants per step per node = 1
- Update timetable = 0 %
- Request route = 100 %
- Exploration probability = 0.1
- Use travel time differences = 0 %
- Update subpaths = True

This means that we have been searching for values for constant A and B to achieve a stable Routing system that does quickly adapt to provide the shortest routes. We are however not interested in the optimal values for constant A and B, because they vary for different sizes of the network environment. More important is how different values for constant A and B affect the stability and the ability to adapt. First we have done some series for varying values of constant A and an unvarying value for constant B. Constant A was subsequently set to 1, 2, 4, 8, 16, 32 and 64, and constant B was set to 0.01. Then we have executed some series where constant A was kept to 4 and constant B was set to 0, 0.005, 0.02 and 0.04. All series consisted of 6 runs.

6.3.3 Expectations

In most cases the first vehicles will take the northern route and after some time new vehicles will follow the southern route. That is, if the parameters are well set and the Routing system works fine. To see the effect of the parameters constant A and B we recall the update function for the probability of the chosen path:

$$\Delta p = \frac{A}{t} + B$$

Δp represents the increase of the probability for going to a certain next node for some destination. So both constant A and constant B can be increased to achieve bigger updates. And lower values for constant A and B will cause smaller updates. The influence of constant A depends on the travel time t of the ants. The influence of constant B solely depends on the number of ants that take a certain route. High values for both constants will cause quicker adaptation but more chance for instability. Low values may prevent or delay the Routing system from adapting. The value for constant B is expected to be responsible for extra randomness in the course of the value for the probability. Notice that even if the update of the probability is small from one ant, many ants can have a large enumerated effect on the probability. So if many ants take the longer route, the probability for this route will increase.

6.3.4 Results

The resulting graphs from experiment 3 can be found in appendix B.3. Here we will only show the most typical graphs. When looking at the graphs, one should bear in mind that node 4 is the best choice for the traffic, so in the best situation the darkest line is the highest of all. Figure 83 shows that for low values of constant A the probabilities change slowly. But even then there is no guarantee that a reached optimal situation is stable. Notice that the probability for node 1 slowly decreases to almost zero. There is no positive update at all. The reason is simple: if ants choose node 1 as their next node, they can only reach their destination node (6) by returning to node 2. In that case they have created a circular path, which will be removed from their memory. So when they return from their destination back to the source node (2) as a backward ant they will never give a positive update to the probability for node 1. Hence all the updates for this probability are negative, as a consequence of positive updates for the other probabilities. The only reason that the probability does not actually reach zero, is because of the use of the exploration probability. No probability can be diminished to a value below the exploration probability divided by the number of neighbouring nodes. In this case the exploration probability has a value of 0.10, so the probability for node 1 will get to 0.033 after some time. This will be the same for all runs of this experiment.

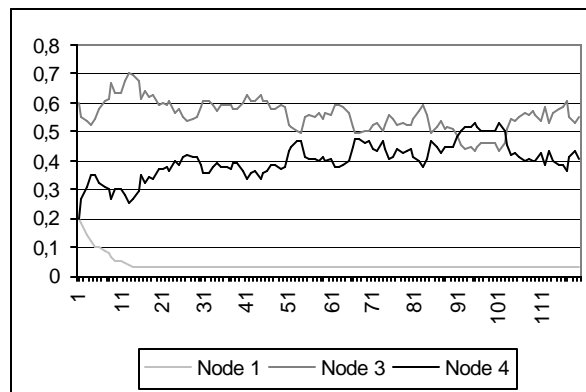


Figure 83: Probabilities for destination 6 ($A = 1$, $B = 0.01$)

The graph of Figure 84 shows more desirable results. The Routing system adapts much quicker to the best situation. And also the system is rather stable after reaching this situation.

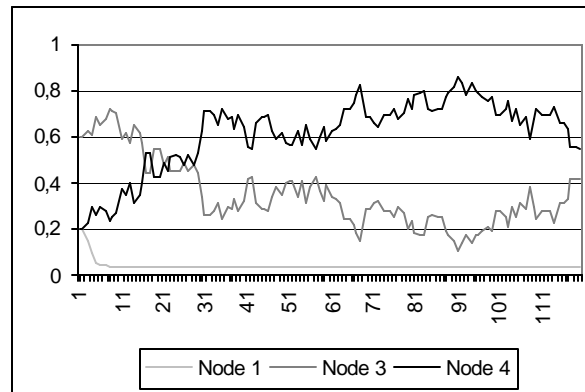


Figure 84: Probabilities for destination 6 ($A = 4$, $B = 0.01$)

In Figure 85 the value for constant A was rather high: 32. This causes the probabilities to change with big jumps. Most of the time the Routing system provides the best route, but probabilities can change from over 0.9 to less than 0.1 in less than 10 seconds.

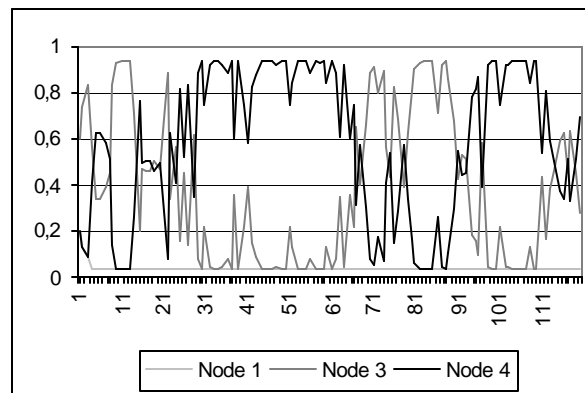


Figure 85: Probabilities for destination 6 ($A = 32$, $B = 0.01$)

Figure 86 shows that the use of constant B has no significant effect on the performance of the ABC - algorithm. Disabling the use of constant B (setting it to zero) results in a performance that is comparable to the case when it was set to 0.01, as shown in Figure 84. The Routing system quickly adapts and the best probability reaches an even higher value, which makes it more stable in the long run.

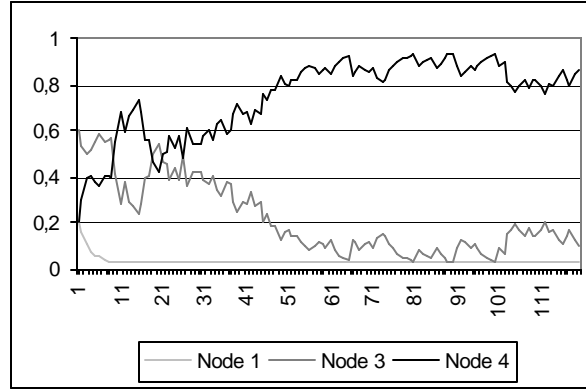


Figure 86: Probabilities for destination 6 ($A = 4$, $B = 0$)

As shown in Figure 87 high values for constant B cause the Routing system to produce rather random routes. This might be an advantage in some cases, when looking at the general traffic distribution. But of course in most cases it results in a poor performance, especially for individual drivers.

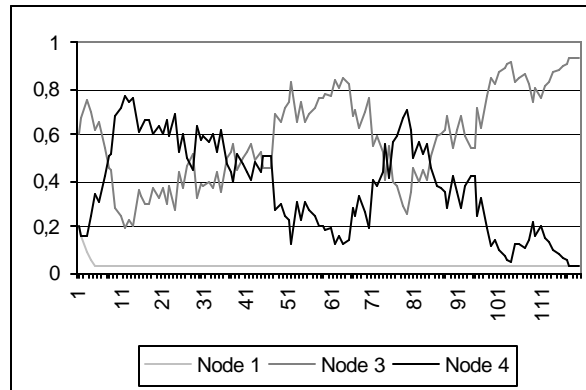


Figure 87: Probabilities for destination 6 ($A = 4$, $B = 0.04$)

We can conclude that the ABC-algorithm still routes some vehicles along the longer paths in time, even with optimal parameter settings. The Routing system will rarely send vehicles via irrational roads to get to their destination. Although in some cases shorter routes may be possible. This results in a very good usage of the network capacity, even though the results for individual users may not always be optimal.

Chapter 7: Conclusions and recommendations

In this last chapter we will discuss the results of our study and draw some conclusions. Furthermore we will present some suggestions for extension and enhancement in the future.

7.1 Conclusions

In this report we have examined the feasibility of a real-time route planner based on dynamic traffic information. Many static route planners are available, but they are not capable of dealing with congestions in cities and on highways. We have investigated the application of Ant Based Control for dynamic vehicle routing. Next to that we have explored the possibilities for gathering real-time traffic information from individual vehicles. We assumed that the position of vehicles is tracked with GPS and this information is communicated to a routing system with GSM or GPRS. These subjects are modelled and combined in a prototype of a routing system. To analyse the operation of the routing system a simulation environment has been build. This environment is capable of simulating traffic in a city. The artificial drivers are confronted with multi-lane roads and one-way roads. They have to deal with precedence rules on normal crossings, with traffic lights and even roundabouts. This allows for quite good traffic simulation. A lot of settings can be adjusted for very realistic traffic behaviour. A disadvantage of this approach is the effort that is needed to create a city with traffic. To meet these inconveniences a considerable good default value is given to most settings.

From route information provide by vehicles we can compute travel times per link in the traffic network. We have developed an algorithm that takes into account that there can be zero, one or more vehicles providing these data. It also considers the age of the information: older information has less influence to the estimated travel time of a link than new information. The quality of the estimates is sufficient for the routing algorithm to compute the shortest paths, although there may be some unrealistic peaks on roads with traffic lights or significant congestions.

From the dynamic data about the traffic network a routing algorithm computes the shortest routes in time from every node in the network to every node. The routing algorithm makes use of the Ant Based Control algorithm. This algorithm, based on the natural behaviour of ants, has proven to be able to route individual drivers, and to reroute them in cases of congestions or blocked roads. In a short time drivers are advised an alternative route, when the original route would delay the driver unnecessarily. We assume that not all vehicles will use this Routing system, but even the vehicles that do not use it do benefit from the system. Because when some vehicles are routed around the congestion, there will be less congestion for the other vehicles.

When the difference in travel time between two good alternatives is small, the parameters A and B in the update formula of the Ant Based Control algorithm have to be adjusted accurately. They are however dependent on the size of routes. Well-chosen values for constant A and B can result in stable and adaptive routing. Some improvements to the routing algorithm have to be made to guarantee even better result, especially for larger networks with more differences in the size of possible routes. We have introduced two new features in the Ant Based Control algorithm since the version of Van der Put [Van der Put 1999]. The first is the introduction of the possibility to judge the quality of a route not only by the complete travel time, but also the experienced delay. This is one step forward in the quality of the routing algorithm. Another improvement is the updating of sub paths. In the original approach the agents used in the ABC-algorithm only updated complete paths. We let the agents also update sub paths of the complete paths. This way the agents use their knowledge more efficient. This has proven to be a considerable improvement for the routing algorithm.

We can conclude that we have succeeded in building a simulation and a routing system that meet the requirements. The simulation is realistic, sufficiently fast, adaptable and expandable. It is capable of real-time visualisation of the traffic network and the vehicles on a Windows NT/2000 PC. The

Routing system is capable of providing good routes, although no optimal routes can be guaranteed. The speed is tolerable for medium-sized networks. Lasting changes to the network can easily be reproduced in the Routing system and temporary changes, like route disabling, are realised straightforwardly.

7.2 Recommendations

In science most answers arouse even more questions. This also applies to this graduation subject. Every time we realised some features, there were new ideas for improvement. Fortunately, this section provides some excellent opportunities to log the ideas that did not make it to the implementation. The recommendations can very well be worked out within the research of TRAIL, the Netherlands Research School for TRAnsport, Infrastructure and Logistics [TRAIL], headed by Bovy. One of their research programs is Seamless Multimodal Mobility (SMM): the utopia in transportation, where travellers are guided during their journey and never have to wait when changing transportation mode, since a central planner knows their destination and allocates transportation means to handle all demands.

7.2.1 Simulation environment

Speed variation

The most unrealistic feature of the vehicles in the simulation environment is currently that they all have the same speed on the same link. While in real life the differences between drivers and between vehicles may be a cause for congestions or delay, this will not happen in our simulation. To solve this, some probabilistic variation should be used in the computation of the speed of a vehicle. This variation should be dependent of the vehicle to represent the type of vehicle and the behaviour of its driver. And the speed should vary per time (period), to represent that drivers never keep a constant speed in a constant environment. Furthermore the speed should be slowly decreasing and increasing when breaking or accelerating. This creates another challenge: the minimum distance to a predecessor or crossing must depend on the speed (of both vehicles). Especially the solution for deceleration and acceleration may be a time consuming process and the necessity should be well considered.

Traffic lights

For the control of the traffic lights we presently use fixed time control. This means that the traffic lights are green or red according to a given (fixed) time period. So the presence of vehicles in front of a traffic light does not influence the colour of the light. In real life this is often not the case. For example the lights for the main road are often green as long as no traffic is detected. So when a vehicle does approach the traffic light, the driver will not have to wait for the light to turn green. Or the green time of traffic lights is extended when there is still traffic detected. Adding these features would improve the traffic flow at intersections with traffic lights. It is however recommended to use fairly good default values for such a control, because it would be unfeasible to copy the exact control mechanism for every crossing with traffic lights.

7.2.2 Timetable updating system

A current disadvantage of the Timetable updating system is the periodic updating. If the vehicles send update information with a large interval, they have travelled several roads and the derived travel times per road are less accurate. If the vehicles update very quickly there is a big chance that they did not move at all, for example because were waiting for the traffic lights to turn green. But when the movement in some time is zero, the computed travel time for the entire link would be infinity. Because when you are not move you will never complete the link. In our model we assume that the vehicle moves at least 1 cm to avoid a division by zero. But then some very high estimates for the travel time are computed. In most cases these will be unfair and the result may be that vehicles will be routed via

a longer alternative. One can think of two reasonable alternative methods. The vehicles should send update information at the end of each link. Or the vehicles should send update information after having covered a fixed distance.

7.2.3 Routing algorithm

Probability update formula

We have realised that we cannot judge the quality of a route only by looking at the complete travel time. The shortest route between two very distant points will still have a large travel time, but it should receive a big update. A first step to overcome this problem has been made. We have made the size of the update not only inversely proportional to the size of the travel time, but the update is also dependent on the experienced delay of the artificial ants. An improvement for the update formula is to keep a history of paths found by the ants together with the travel times. Then newly found paths should be compared to formerly found paths and the size of the update should depend on whether the new path is shorter or longer. See [Di Caro 1998] for their approach in this matter.

Hierarchical routing

The size of the network is a limiting factor for the ABC algorithm. The algorithm does not perform as well for big networks as for small networks. To overcome this problem a hierarchy of network could be introduced. In that case all main roads and intersections should be part of a network. And separate networks should be used to route in smaller districts. Then a vehicle will first be routed via the main roads and when it gets in the neighbourhood of its destination, the routing is done with the aid of the district network. This way the network can be smaller and the routing algorithm will be more effective. This approach can be extended with more levels, like a network with highways, or even international routing. One should bear in mind that there is a risk that alternatives on a lower level may be missed when routing on a higher level. For example when there is congestion on a highway, a good alternative may be to go off the highway. Such an alternative will not be offered when routing on the highway level.

Routing to closest car park

We have used the ABC algorithm to route vehicles from a given location to another given location. But in some cases a driver may not be interested in or may not know the exact destination. For example, the user wants to go to the town centre. He does not know where to park, but he just wants the route to the closest car park from his point of view. So when he comes into town from the south, he wants to go to the most southern car park in the town centre. And when he enters the town from the north, he wishes to be routed to the most northern car park in the town centre. So his desired destination depends on his current position. For this purpose the ABC algorithm is in fact perfectly suited as opposed to many other routing algorithms. Remember the original application in nature from the real life ants. They use the method to find the shortest way from their nest to a food source. When there are several food sources with the same volume and nutritional value, they just look for the closest (until it is finished). If we now add an identical identification number to the locations with a car park and we let the artificial ants also search for routes to this destination type, the problem is solved. From then on the driver can enter a car park as destination and will automatically be routed to the closest car park. Of course this can also be used for several other purposes, like the closest petrol station, the closest restaurant, the closest shopping mall, etcetera, etcetera.

Appendix A: Bibliography

- [ANWB] ANWB, www.anwb.nl.
- [Bonabeau 1999] Bonabeau E., Dorigo M., Theraulaz G., *Swarm Intelligence, From Natural to Artificial Systems*, Oxford University Press, 1999.
- [Cormen 2000] Cormen T.H., Leiserson C.E., Rivest R.L., *Introduction to Algorithms*, MIT Press, 2000.
- [Dewilde 2002] Dewilde P., *Smart Roads*, Internal Report ITS, Delft University of Technology, 2002.
- [Di Caro 1997] Di Caro G., Dorigo M., *AntNet: A Mobile Agents Approach to Adaptive Routing*, Technical Report 97-12, IRIDIA, Université Libre de Bruxelles, 1997.
- [Di Caro 1998] Di Caro G., Dorigo M., *AntNet: Distributed Stigmergetic Control for Communication Networks*, Journal of Artificial Intelligence Research (JAIR), vol. 9 p. 317-365, 1998.
- [Dorigo 1997] Dorigo M., Gambardella L.M., *Ant Colony System: A Cooperative Learning Approach to the Travelling Salesman Problem*, IEEE Transactions on Evolutionary Computation, vol. 1 no. 1 p. 53-66, 1997.
- [Eggenkamp 2001] Eggenkamp G., *Dynamic multi modal route planning: an artificial intelligence approach*, Master's thesis, Delft University of Technology, 2001.
- [Franks 1989] Franks N.R., *Army Ants: A Collective Intelligence*, American Scientist, vol. 77 March-April, 1989.
- [Knibbe 1999] Knibbe H.P., *Adaptive Routing With Mobile Agents*, Master's thesis, Delft University of Technology, 1999.
- [PROMISE] PROMISE, www.promise.cellulardata.com.
- [Schoonderwoerd 1997] Schoonderwoerd R., Holland O., Bruten J., Rothkrantz L.J.M., *Load Balancing in Telecommunication Networks*, Adaptive Behaviour, vol. 5 no. 2, 1997.
- [Tatomir 2002] Tatomir B., Rothkrantz L.J.M., *Comparison of Ant Based Control Algorithms*, Technical Report DKS-02-02, Delft University of Technology, 2002 (forthcoming).
- [Tegaron] Tegaron Scout, www.tegaron.de.
- [TRAIL] TRAIL, www.rstrail.nl.
- [Travelstar] Travelstar, www.travelstar.nl.

[Van der Put 1999]

Put R. van der, Rothkrantz L.J.M., *Routing in packet switched networks using mobile agents*, Journal of Simulation Practice and Theory, 1999.

Appendix B: Experimental results

B.1 Experiment 1

Environment

This experiment is executed using test4.city for the traffic environment. Before starting the simulation roads 25 and 26 were disabled.

Default parameters

All runs of this experiment have used the following default parameters unless specified otherwise.

- Traffic steps per second = 5
- Ant steps per second = 5
- Generate vehicles = True
- Generate ants = True
- Vehicles per second = 1.0
- Ants per step per node = 1.0
- Update timetable = 50 %
- Request route = 50 %
- Constant A = 0.8
- Constant B = 0.01
- Exploration probability = 0.1
- Travel time differences = 50 %
- Update subpaths = True

Series 1: all parameters as default

In this series none of the default parameters were changed.

Table 9: Series 1 of experiment 1

Run	First time adaptation of smart vehicles
1	126 s
2	122 s
3	348 s
4	183 s *
5	313 s
6	194 s *
7	211 s
8	174 s
9	181 s
10	76 s

Average: 192.8 s

Standard deviation (with n-1): 83.4 s

Series 2: No updating of sub paths

In this series the sub paths are not updated.

Table 10: Series 2 of experiment 1

Run	First time adaptation of smart vehicles
1	572 s *
2	371 s *
3	218 s
4	962 s *
5	121 s
6	401 s *
7	918 s
8	218 s *
9	285 s
10	156 s *

Average: 422.2 s

Standard deviation (with n-1): 303.0 s

Series 3: 2 ants per step per node

In this series 2 ants per step per node are generated (instead of 1).

Table 11: Series 3 of experiment 1

Run	First time adaptation of smart vehicles
1	136 s
2	95 s *
3	46 s
4	213 s
5	129 s
6	67 s
7	73 s
8	224 s *
9	149 s
10	86 s

Average: 121.8 s

Standard deviation (with n-1): 60.4 s

Series 4: 10 ant steps per second

In this series 10 ant steps per second are done (instead of 5).

Table 12: Series 4 of experiment 1

Run	First time adaptation of smart vehicles
1	63 s *
2	34 s
3	25 s
4	91 s
5	79 s
6	110 s
7	333 s
8	118 s *
9	73 s *
10	70 s *

Average: 99.6 s

Standard deviation (with n-1): 87.1 s

Series 5: Constant A = 0.6

In this series constant A is set to 0.6 (instead of 0.8).

Table 13: Series 5 of experiment 1

Run	First time adaptation of smart vehicles
1	326 s
2	107 s *
3	202 s *
4	173 s *
5	93 s
6	254 s
7	296 s *
8	152 s
9	123 s
10	123 s

Average: 184.9 s

Standard deviation (with n-1): 82.1 s

Series 6: Constant A = 1.2

In this series constant A is set to 1.2 (instead of 0.8).

Table 14: Series 6 of experiment 1

Run	First time adaptation of smart vehicles
1	357 s
2	73 s
3	117 s *
4	126 s *
5	96 s
6	364 s
7	83 s
8	67 s
9	121 s
10	175 s

Average: 157.9 s

Standard deviation (with n-1): 111.2 s

Series 7: Constant A = 1.6

In this series constant A is set to 1.6 (instead of 0.8).

Table 15: Series 7 of experiment 1

Run	First time adaptation of smart vehicles
1	143 s
2	123 s
3	108 s
4	252 s
5	125 s
6	55 s
7	176 s
8	74 s
9	83 s
10	128 s *

Average: 126.7 s

Standard deviation (with n-1): 56.4 s

Series 8: Constant A = 2.0

In this series constant A is set to 2.0 (instead of 0.8).

Table 16: Series 8 of experiment 1

Run	First time adaptation of smart vehicles
1	136 s
2	155 s
3	42 s *
4	307 s
5	32 s
6	72 s
7	139 s *
8	89 s
9	156 s
10	162 s

Average: 129.0 s

Standard deviation (with n-1): 79.0 s

Series 9: Constant A = 3.0

In this series constant A is set to 3.0 (instead of 0.8).

Table 17: Series 9 of experiment 1

Run	First time adaptation of smart vehicles
1	34 s
2	66 s
3	57 s *
4	70 s
5	52 s
6	52 s
7	177 s
8	67 s
9	158 s
10	44 s

Average: 77.7 s

Standard deviation (with n-1): 48.8 s

Series 10: Constant A = 4.0

In this series constant A is set to 4.0 (instead of 0.8).

Table 18: Series 10 of experiment 1

Run	First time adaptation of smart vehicles
1	115 s
2	34 s
3	71 s
4	44 s
5	50 s
6	21 s
7	50 s
8	78 s
9	199 s *
10	55 s

Average: 71.7 s

Standard deviation (with n-1): 51.7 s

Series 11: Constant A = 5.0

In this series constant A is set to 5.0 (instead of 0.8).

Table 19: Series 11 of experiment 1

Run	First time adaptation of smart vehicles
1	62 s
2	41 s *
3	25 s
4	56 s *
5	122 s
6	60 s
7	64 s
8	61 s
9	65 s *
10	77 s

Average: 63.3 s

Standard deviation (with n-1): 25.1 s

Series 12: Constant A = 6.0

In this series constant A is set to 6.0 (instead of 0.8).

Table 20: Series 12 of experiment 1

Run	First time adaptation of smart vehicles
1	34 s
2	26 s
3	57 s
4	22 s
5	44 s
6	52 s
7	31 s
8	68 s *
9	32 s
10	53 s *

Average: 41.9 s

Standard deviation (with n-1): 15.2 s

Series 13: Constant A = 7.0

In this series constant A is set to 7.0 (instead of 0.8).

Table 21: Series 13 of experiment 1

Run	First time adaptation of smart vehicles
1	34 s
2	60 s *
3	84 s
4	37 s
5	138 s *
6	21 s *
7	37 s
8	41 s
9	61 s
10	82 s *

Average: 59.5 s

Standard deviation (with n-1): 34.6 s

Series 14: Constant A = 8.0

In this series constant A is set to 8.0 (instead of 0.8).

Table 22: Series 14 of experiment 1

Run	First time adaptation of smart vehicles
1	74 s
2	93 s
3	26 s
4	11 s *
5	30 s *
6	20 s
7	44 s *
8	141 s
9	11 s
10	161 s

Average: 61.1 s

Standard deviation (with n-1): 54.5 s

Series 15: Constant A = 9.0

In this series constant A is set to 9.0 (instead of 0.8).

Table 23: Series 15 of experiment 1

Run	First time adaptation of smart vehicles
1	85 s
2	70 s *
3	57 s
4	204 s *
5	12 s *
6	60 s
7	75 s *
8	104 s
9	24 s *
10	16 s *

Average: 70.7 s

Standard deviation (with n-1): 55.8 s

* = The state of the Routing system was not stable enough, so some vehicles took the wrong way after the first vehicle had taken a shorter way.

B.2 Experiment 2

Environment

This experiment is executed using test3.city for the traffic environment.

Parameters

For this experiment the parameters are set to following values.

- Seconds = 2400
- Runs = 1
- Traffic steps per second = 5
- Ant steps per second = 5
- Generate vehicles = True
- Generate ants = True
- Vehicles per second = 1.0
- Ants per step per node = 1.0
- Update timetable = 50 %
- Request route = 50 %
- Constant A = 0.8
- Constant B = 0.01
- Exploration probability = 0.1
- Travel time differences = 50 %
- Update subpaths = True

Results

In this table the average travel times per road are shown, which were computed over a period of 2400 seconds.

Table 24: Travel times per road for experiment 2

Road number	From node	To node	Length	Minimum travel time	Average travel time
1	1	2	320 m	23.04 s	28.70 s
2	2	1	320 m	23.04 s	22.98 s
3	2	3	330 m	23.76 s	23.39 s
4	3	2	330 m	23.76 s	23.98 s
5	3	4	80 m	5.76 s	5.86 s
6	4	3	80 m	5.76 s	5.96 s
7	2	6	358 m	25.78 s	50.28 s
8	6	2	358 m	25.78 s	26.69 s
9	5	6	320 m	23.04 s	70.58 s
10	6	5	320 m	23.04 s	n/a
11	6	7	320 m	23.04 s	n/a
12	7	6	320 m	23.04 s	71.80 s
13	4	8	80 m	5.76 s	5.85 s
14	8	4	80 m	5.76 s	5.98 s
15	8	9	330 m	23.76 s	24.93 s
16	9	8	330 m	23.76 s	23.09 s
17	6	9	358 m	25.78 s	25.64 s
18	9	6	358 m	25.78 s	54.78 s
19	9	10	320 m	23.04 s	23.02 s
20	10	9	320 m	23.04 s	25.10 s

B.3 Experiment 3

Environment

This experiment is executed using test5.city for the traffic environment.

Default parameters

All runs of this experiment have used the following parameters. Only constants A and B where changed per series.

- Seconds = 120
- Runs = 1
- Traffic steps per second = 5
- Ant steps per second = 5
- Generate vehicles = True
- Generate ants = True
- Vehicles per second = 0.25
- Ants per step per node = 1
- Update timetable = 0 %
- Request route = 100 %
- Exploration probability = 0.1
- Travel time differences = 50 %
- Update subpaths = True

Results

For all runs of this experiment a graph is made from some values in the probability table of node 2. It concerns the probabilities for destination node 6 and all neighbouring nodes (1, 3, 4).

Series 1: constant $A = 1$, constant $B = 0.01$

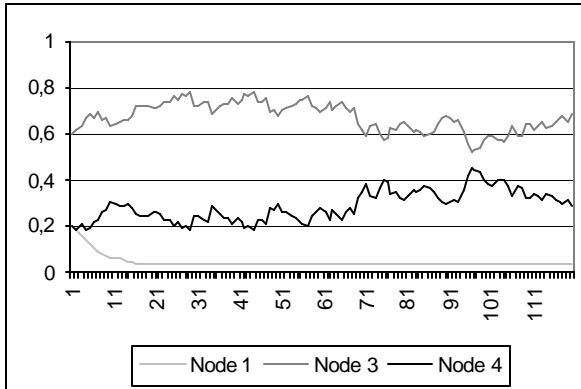


Figure 88: Run 1

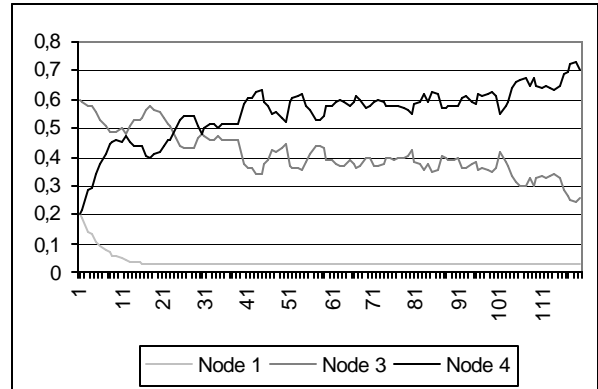


Figure 89: Run 2

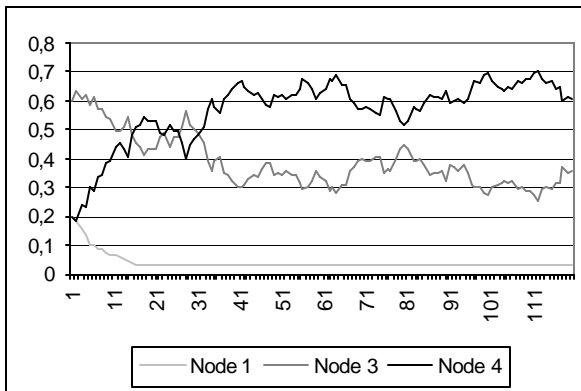


Figure 90: Run 3

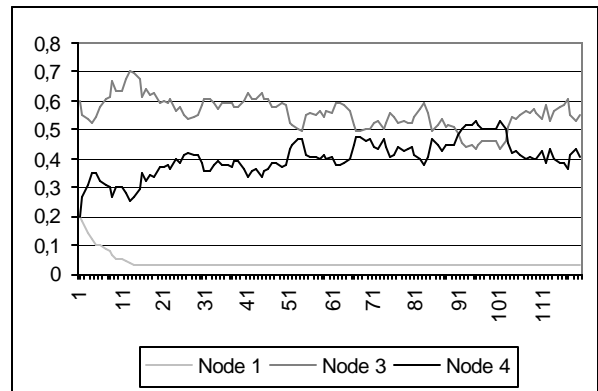


Figure 91: Run 4

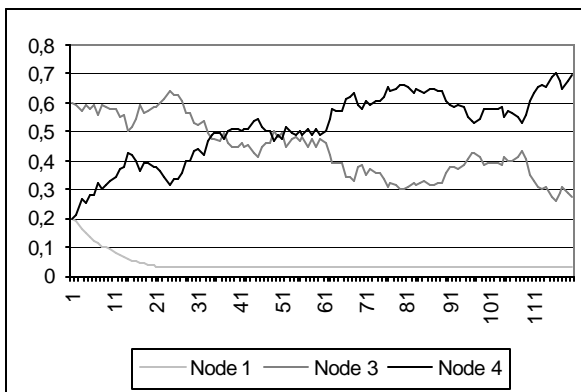


Figure 92: Run 5

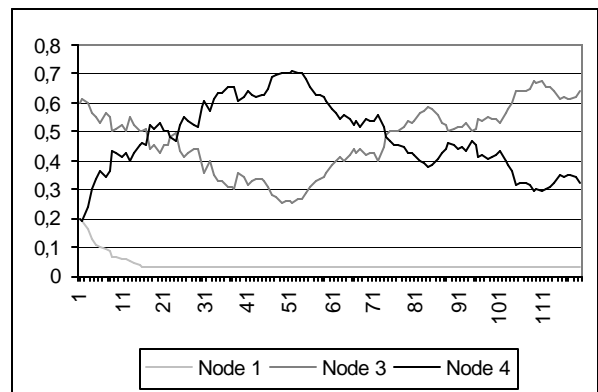


Figure 93: Run 6

Series 2: constant $A = 2$, constant $B = 0.01$

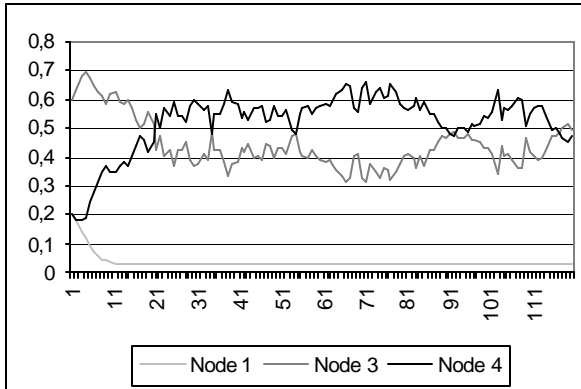


Figure 94: Run 1

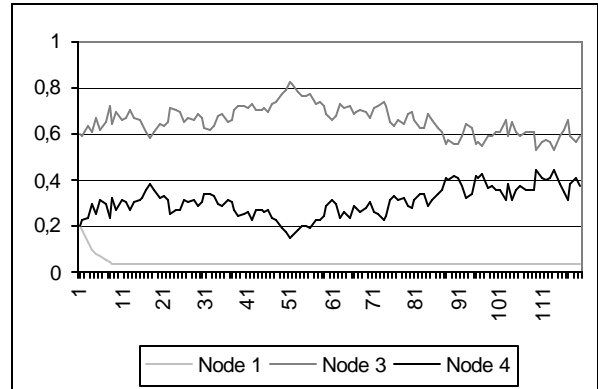


Figure 95: Run 2

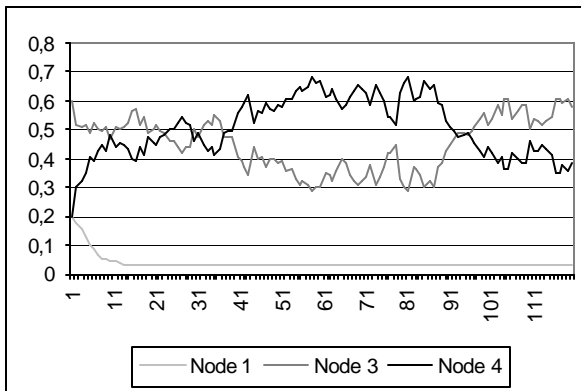


Figure 96: Run 3

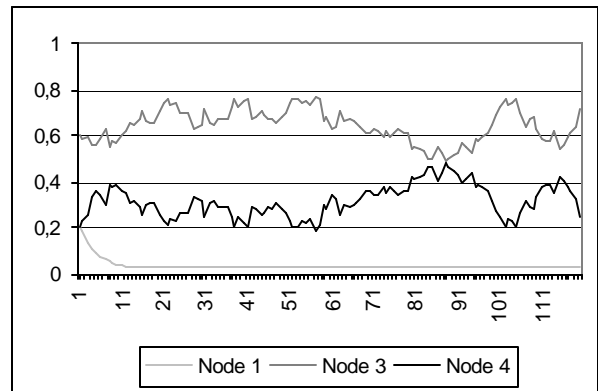


Figure 97: Run 4

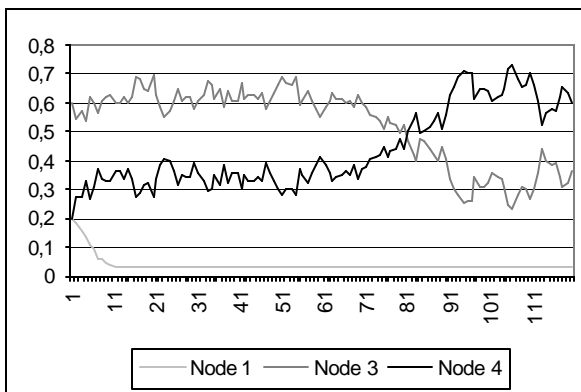


Figure 98: Run 5

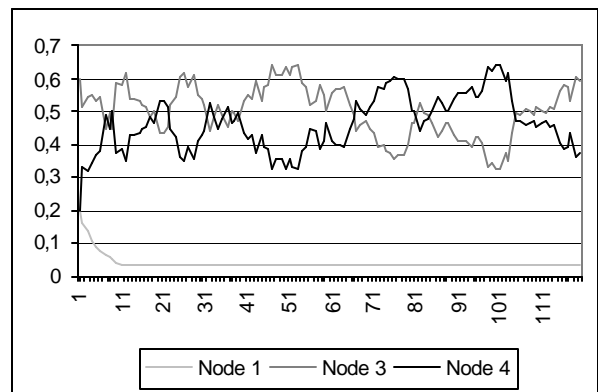


Figure 99: Run 6

Series 3: constant $A = 4$, constant $B = 0.01$

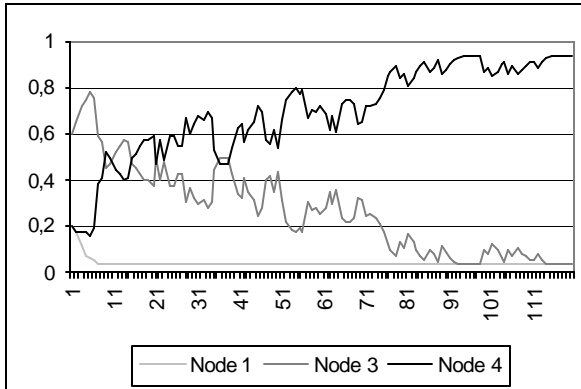


Figure 100: Run 1

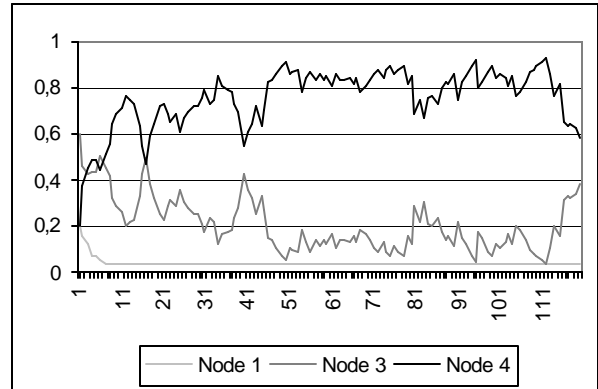


Figure 101: Run 2

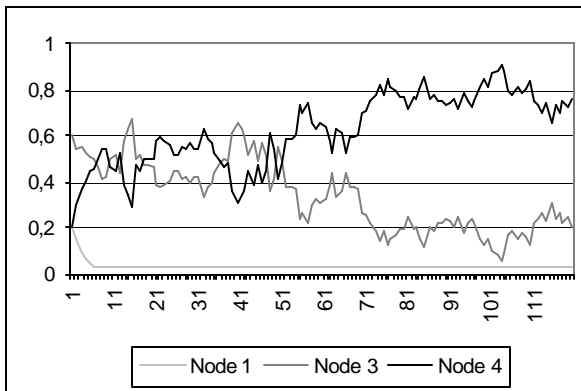


Figure 102: Run 3

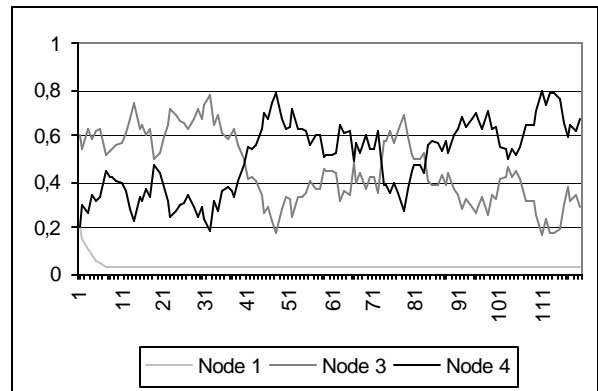


Figure 103: Run 4

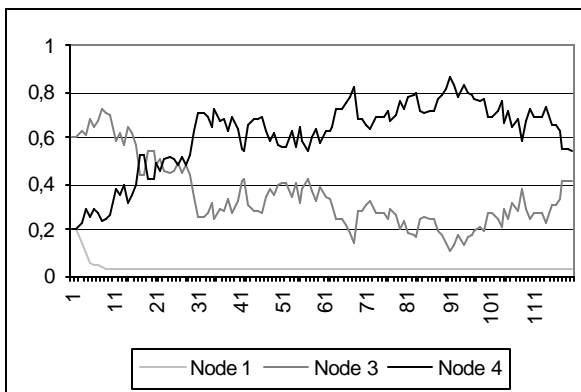


Figure 104: Run 5

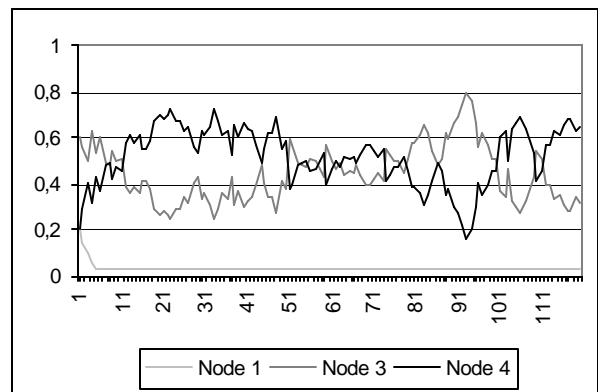


Figure 105: Run 6

Series 4: constant $A = 8$, constant $B = 0.01$

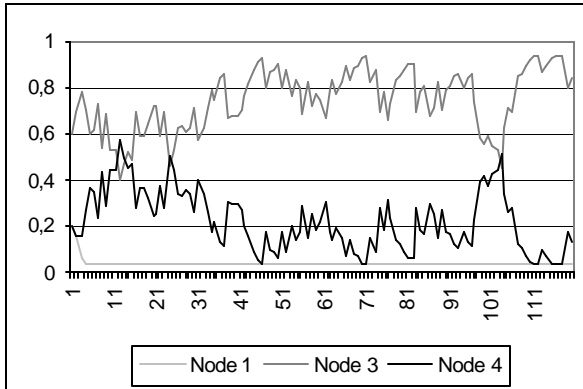


Figure 106: Run 1

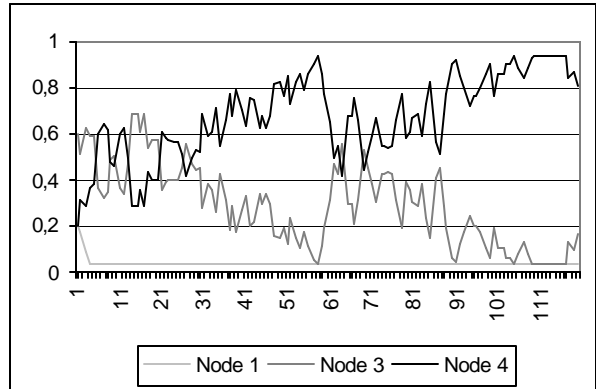


Figure 107: Run 2

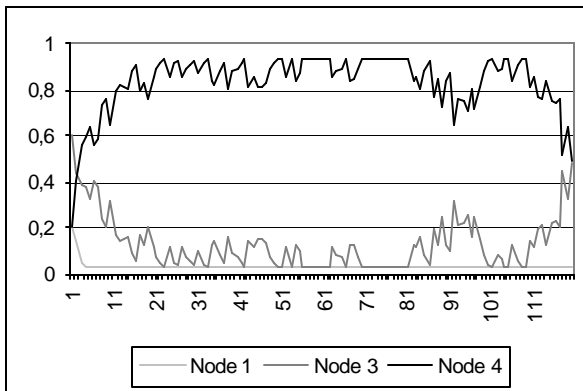


Figure 108: Run 3

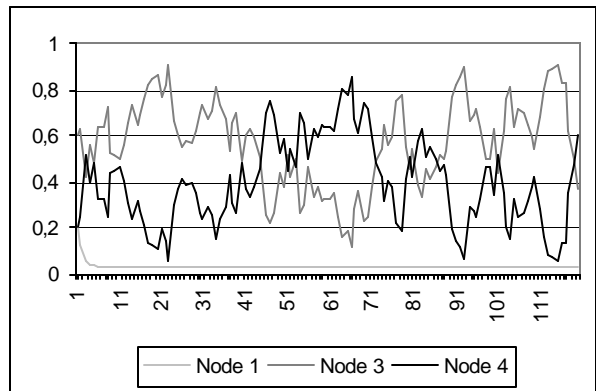


Figure 109: Run 4

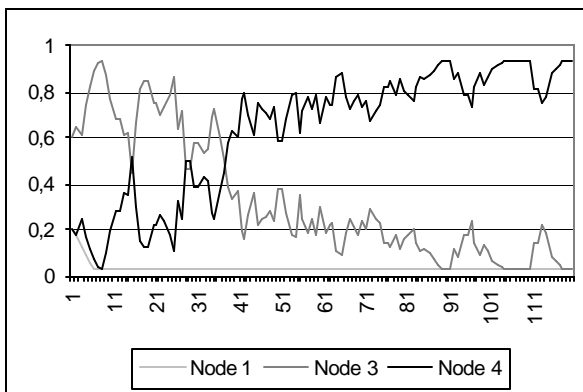


Figure 110: Run 5

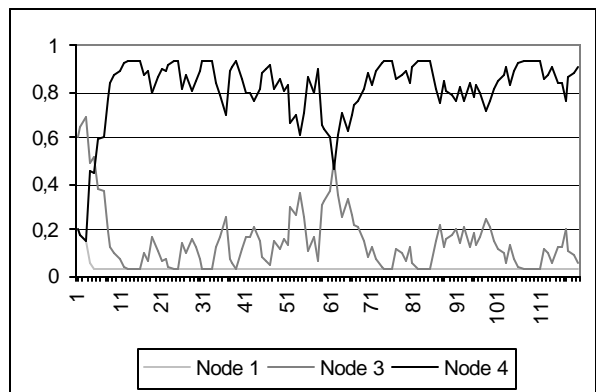


Figure 111: Run 6

Series 5: constant $A = 16$, constant $B = 0.01$

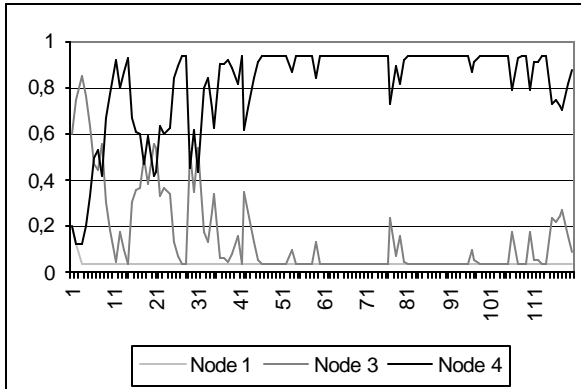


Figure 112: Run 1

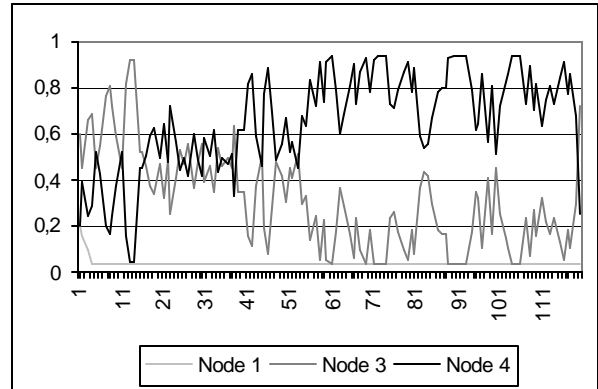


Figure 113: Run 2

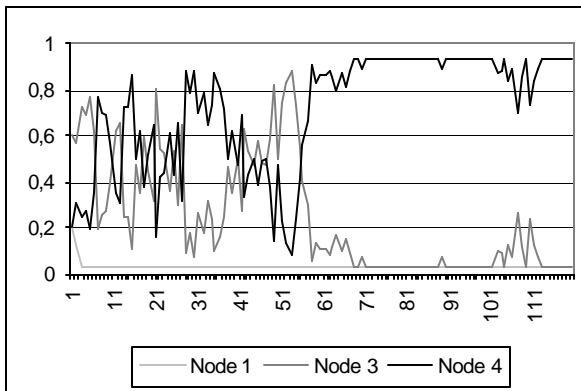


Figure 114: Run 3

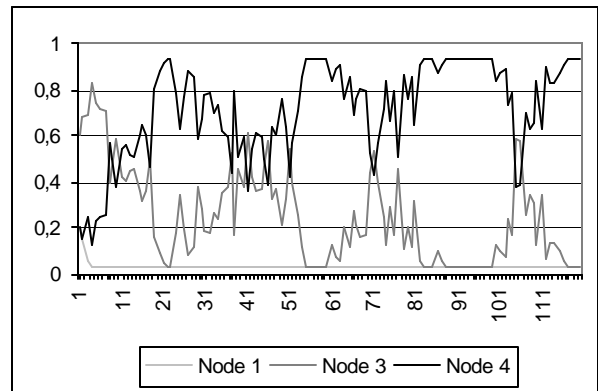


Figure 115: Run 4

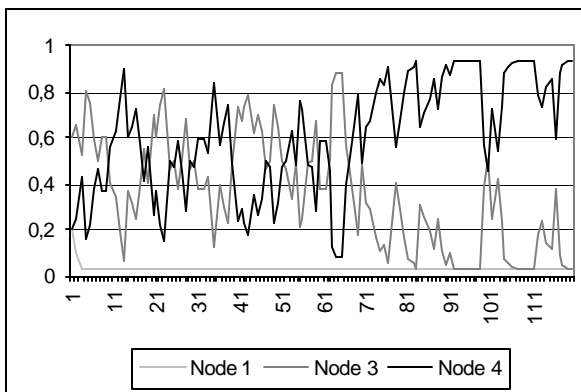


Figure 116: Run 5

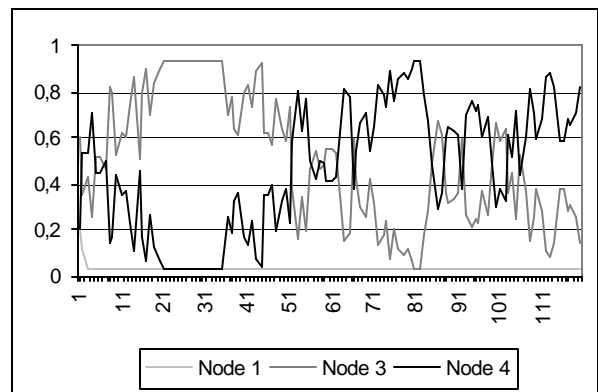


Figure 117: Run 6

Series 6: constant $A = 32$, constant $B = 0.01$

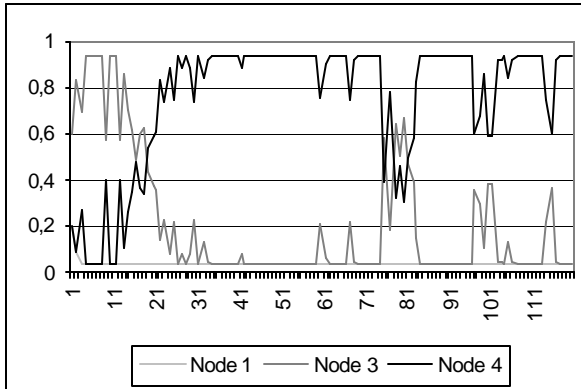


Figure 118: Run 1

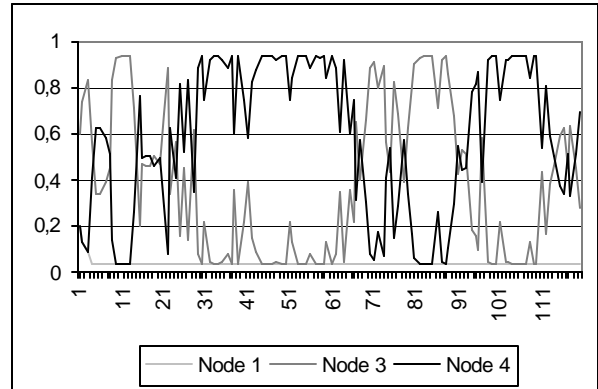


Figure 119: Run 2

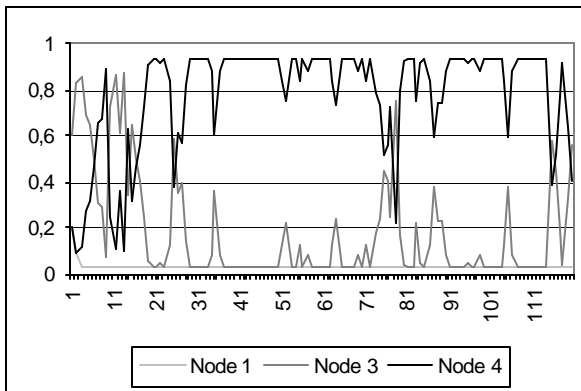


Figure 120: Run 3

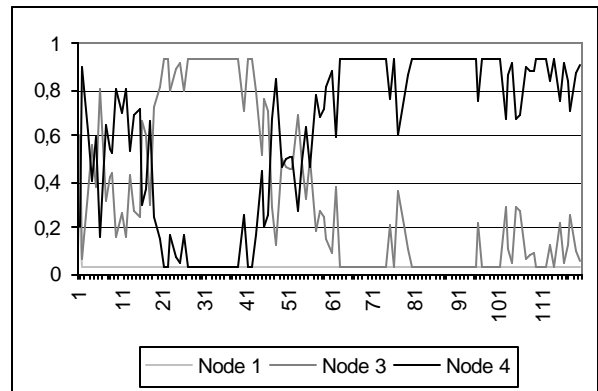


Figure 121: Run 4

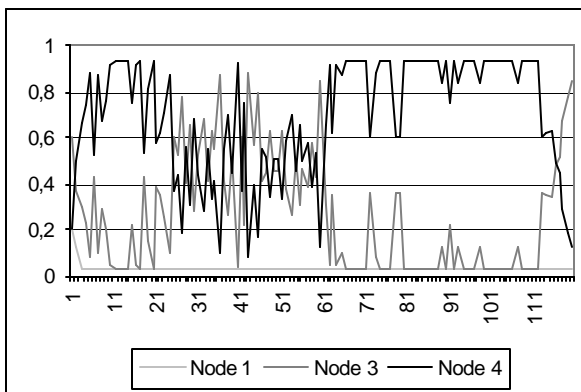


Figure 122: Run 5

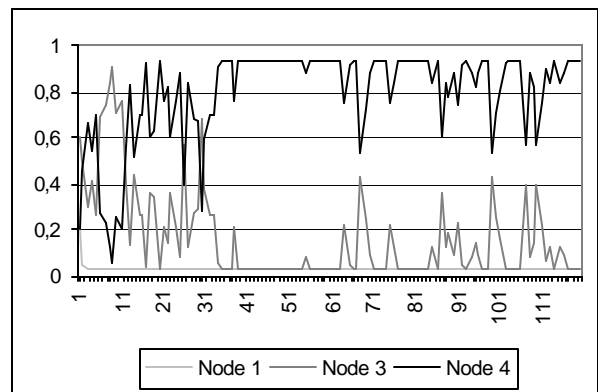


Figure 123: Run 6

Series 7: constant $A = 64$, constant $B = 0.01$

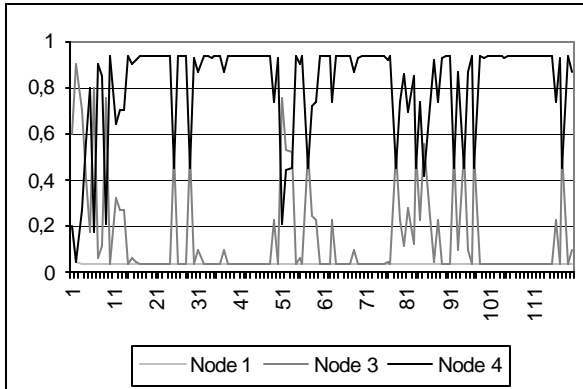


Figure 124: Run 1

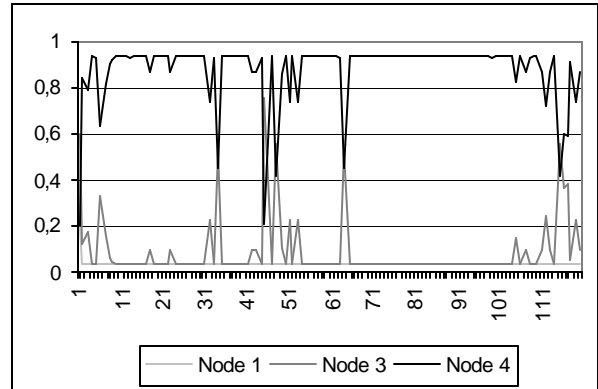


Figure 125: Run 2

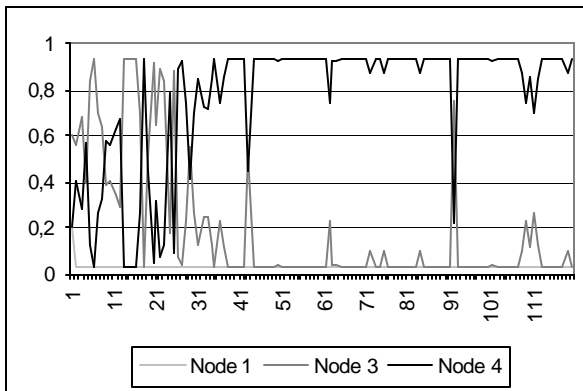


Figure 126: Run 3

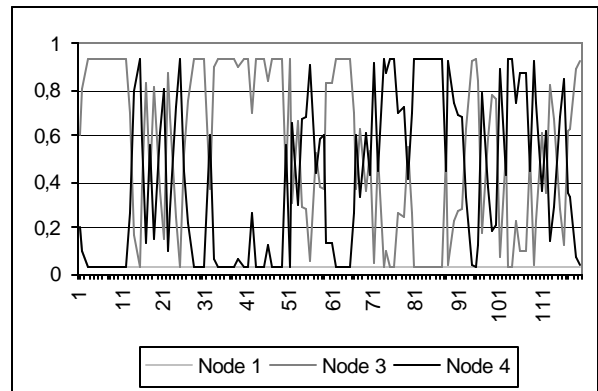


Figure 127: Run 4

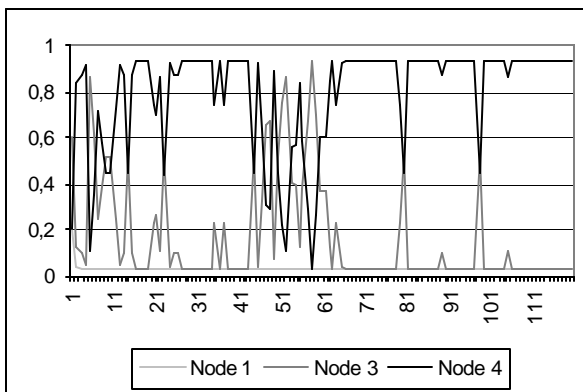


Figure 128: Run 5

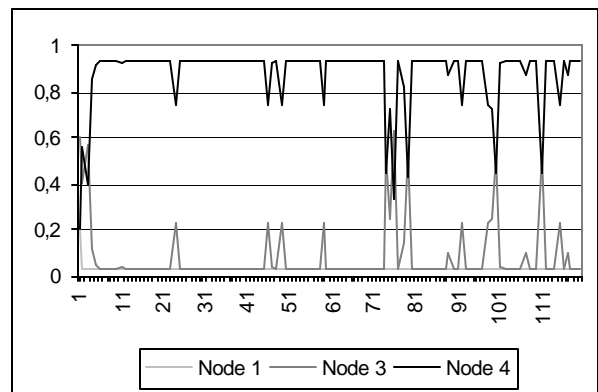


Figure 129: Run 6

Series 8: constant $A = 4$, constant $B = 0$

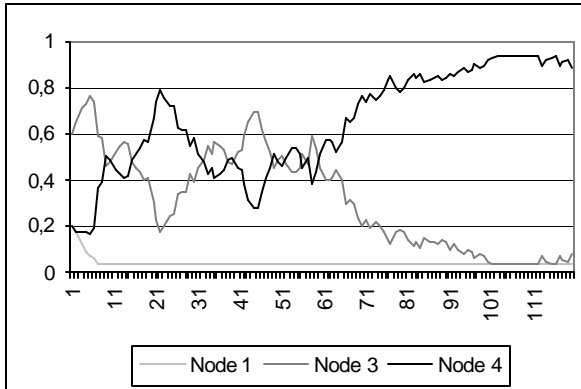


Figure 130: Run 1

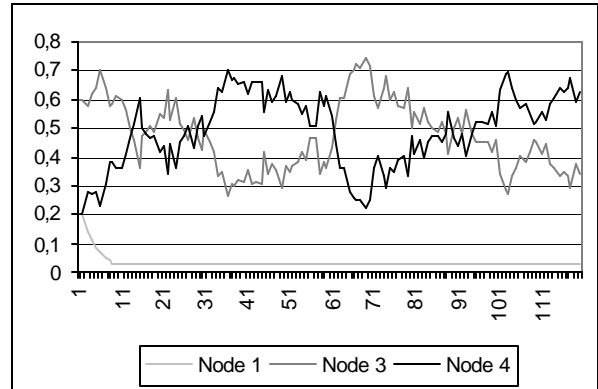


Figure 131: Run 2

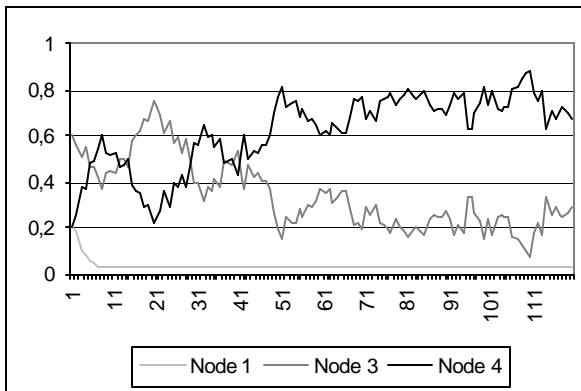


Figure 132: Run 3

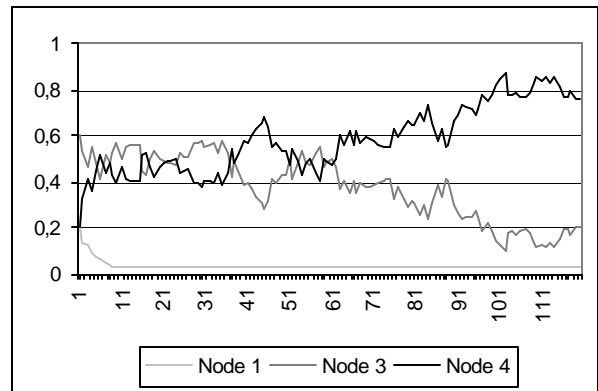


Figure 133: Run 4

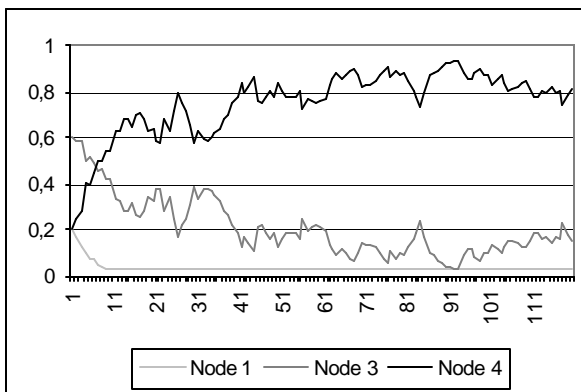


Figure 134: Run 5

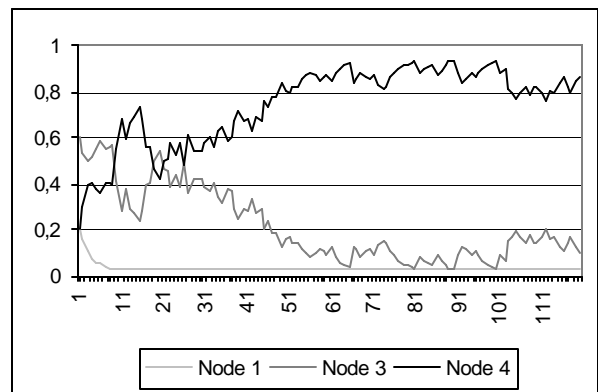


Figure 135: Run 6

Series 9: constant $A = 4$, constant $B = 0.005$

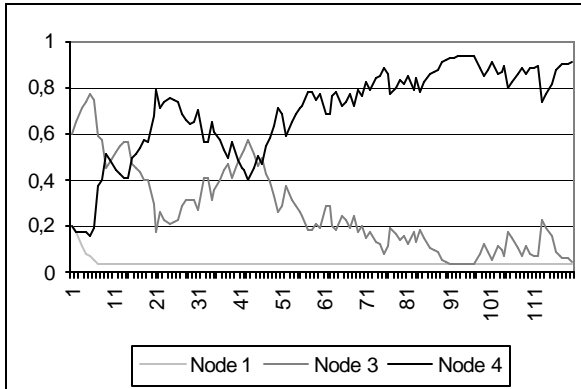


Figure 136: Run 1

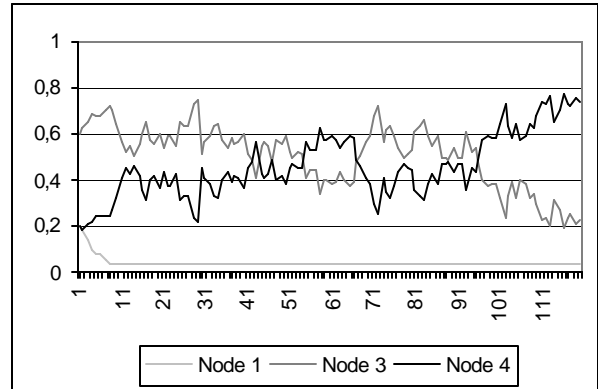


Figure 137: Run 2

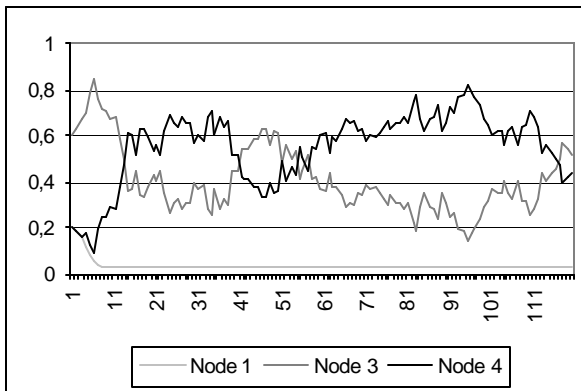


Figure 138: Run 3

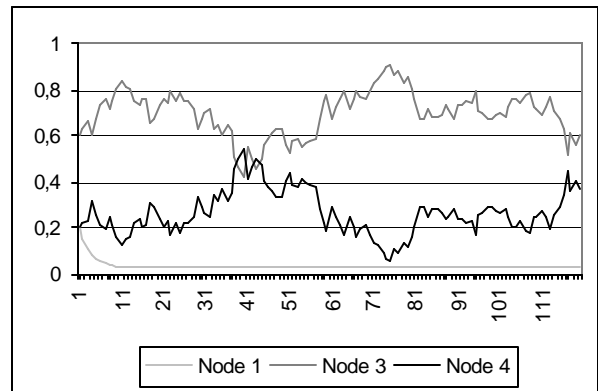


Figure 139: Run 4

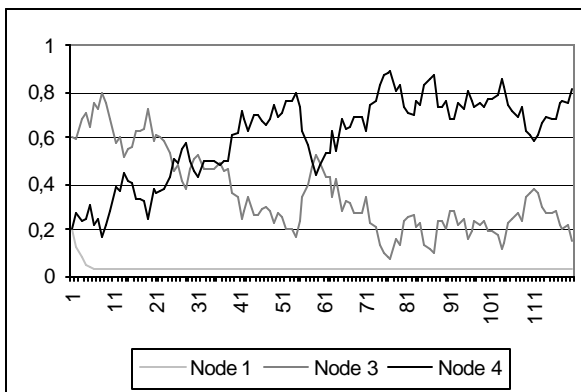


Figure 140: Run 5

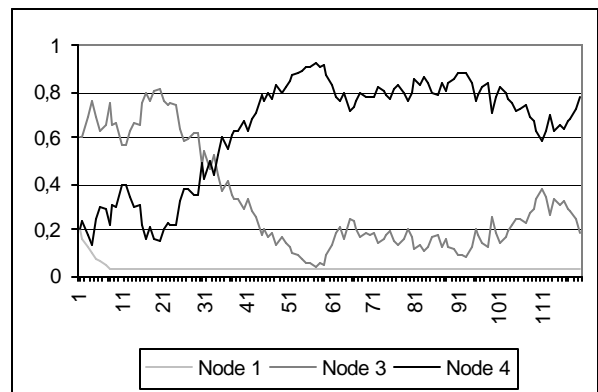


Figure 141: Run 6

Series 10: constant $A = 4$, constant $B = 0.02$

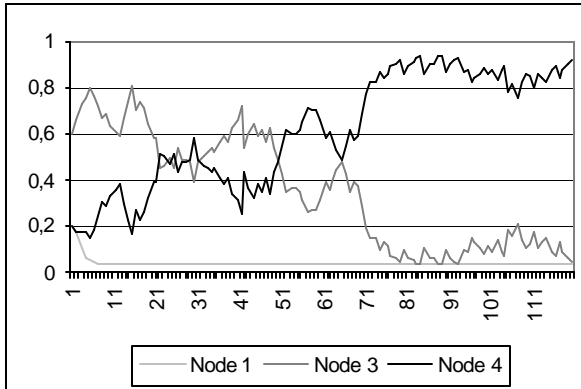


Figure 142: Run 1

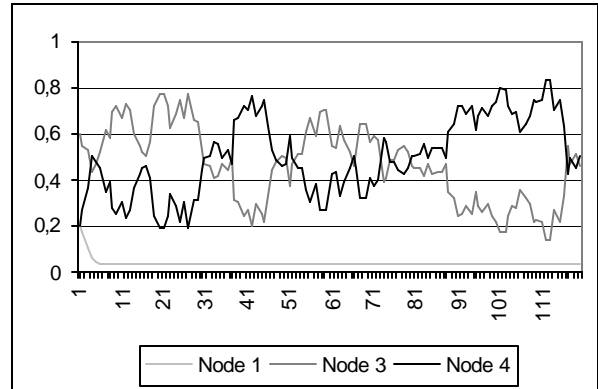


Figure 143: Run 2

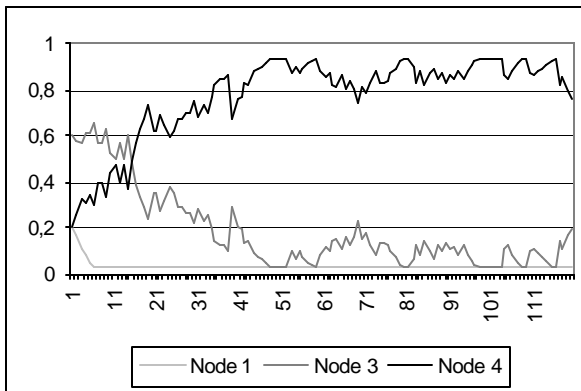


Figure 144: Run 3

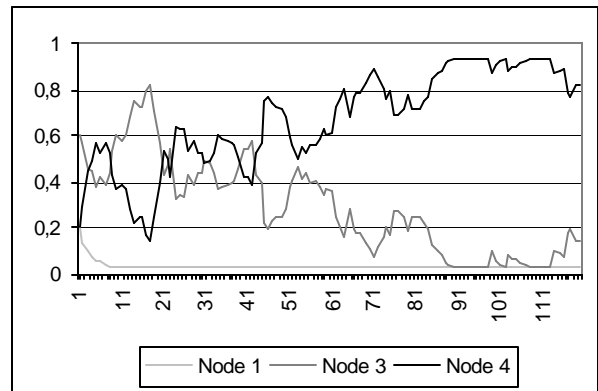


Figure 145: Run 4

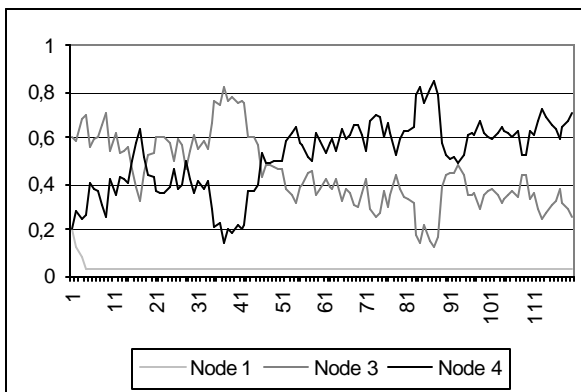


Figure 146: Run 5

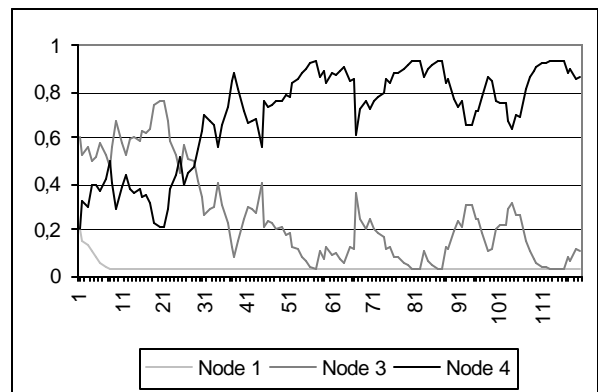


Figure 147: Run 6

Series 11: constant $A = 4$, constant $B = 0.04$

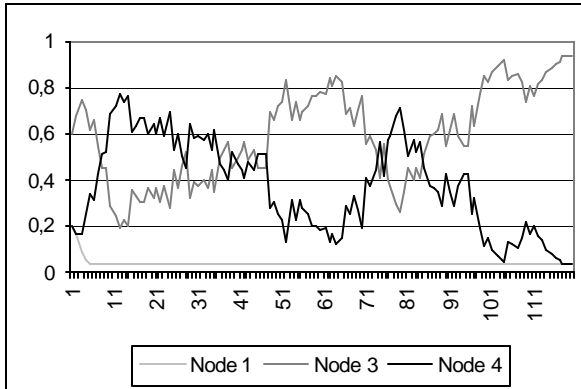


Figure 148: Run 1

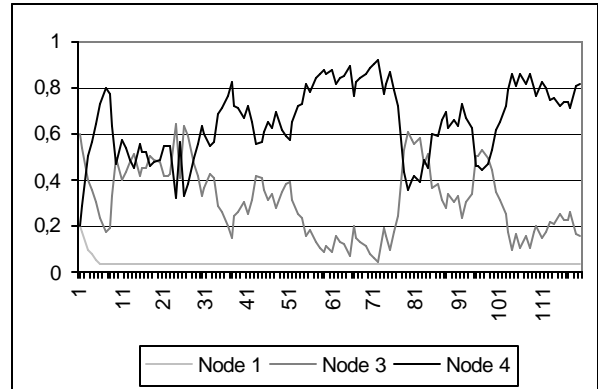


Figure 149: Run 2

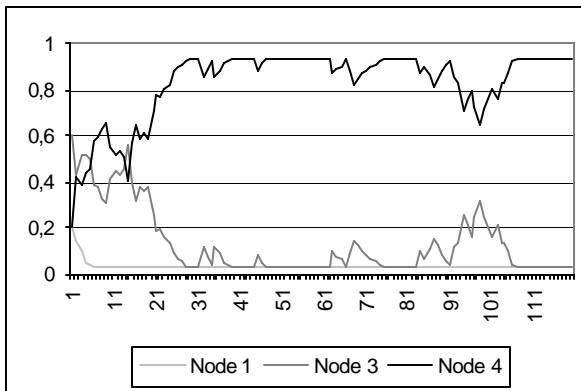


Figure 150: Run 3

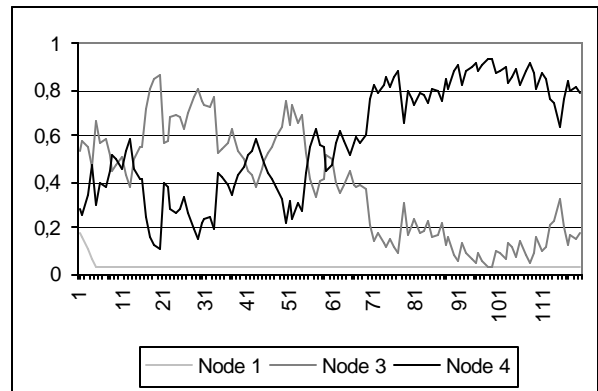


Figure 151: Run 4

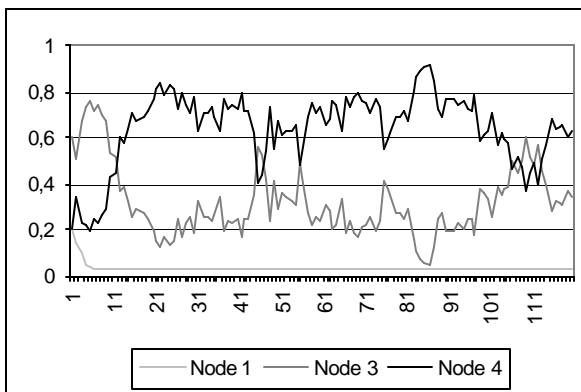


Figure 152: Run 5

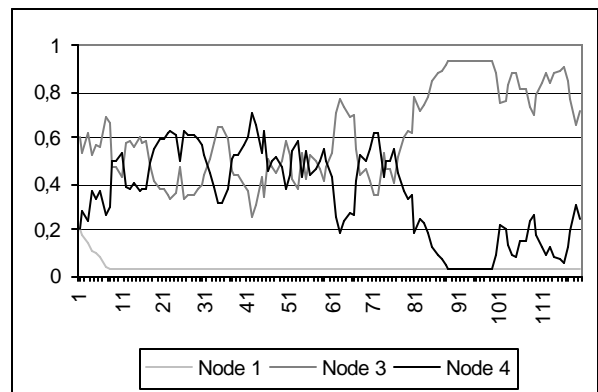


Figure 153: Run 6

Appendix C: User manual

Here we will explain the operation of the system. From the design we know that our system is composed of two basic modules. These are the programs 'City.exe' and 'RoutingSystem.exe'. 'City.exe' is a simulation of a traffic environment. The user can load one of the environments existing in our database or he can add his own environment. 'RoutingSystem.exe' is program used to route the vehicles in the traffic environment. First we will show the basic steps needed to run a simulation with an existing environment. And further more we will explain all the possibilities that the programs provide. This chapter is definitely a must for any researcher that wants to use the system to its fullest extend, but it is also important for anyone who wants to give a simple demonstration.

C.1 The basic steps to operate the system

- The program should be started by running the file 'City.exe'. This would produce Figure 154 or a similar one.

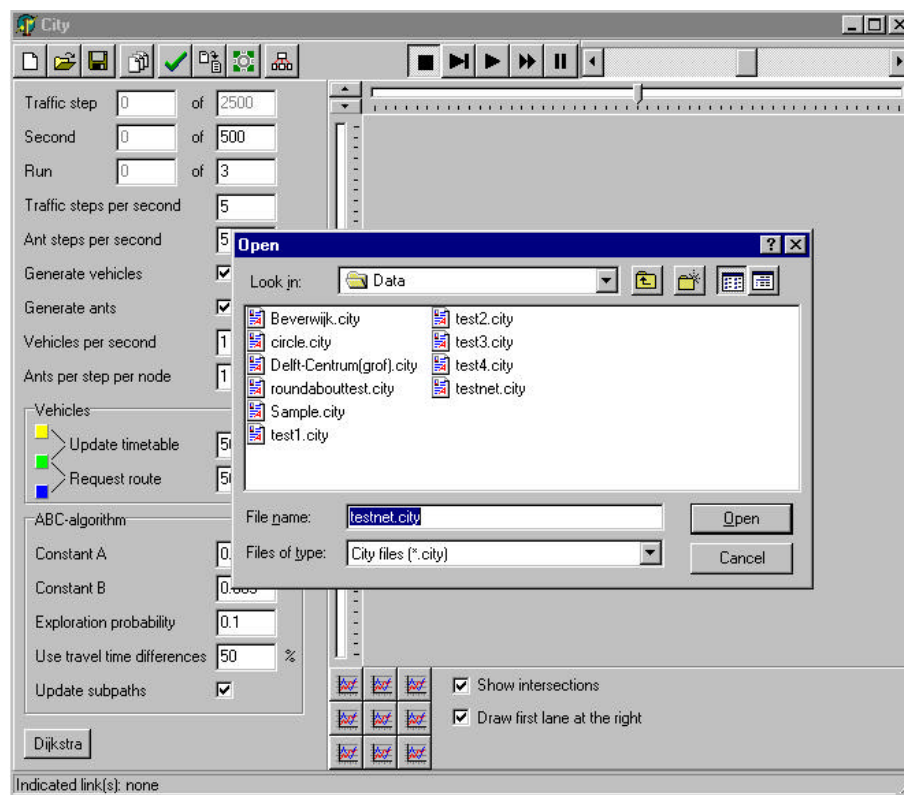


Figure 154: The opening screen from the City program

- The program will automatically popup an 'Open' dialog in the middle of the screen. The rest of the screen is not important yet. Choose a file with a '.city' extension by clicking on the file and choose open. Now the file of your choice will be loaded together with all other needed data that belongs to the file of your choice. Imaging that you chose 'test1.city' then Figure 155 or a similar one will be shown.

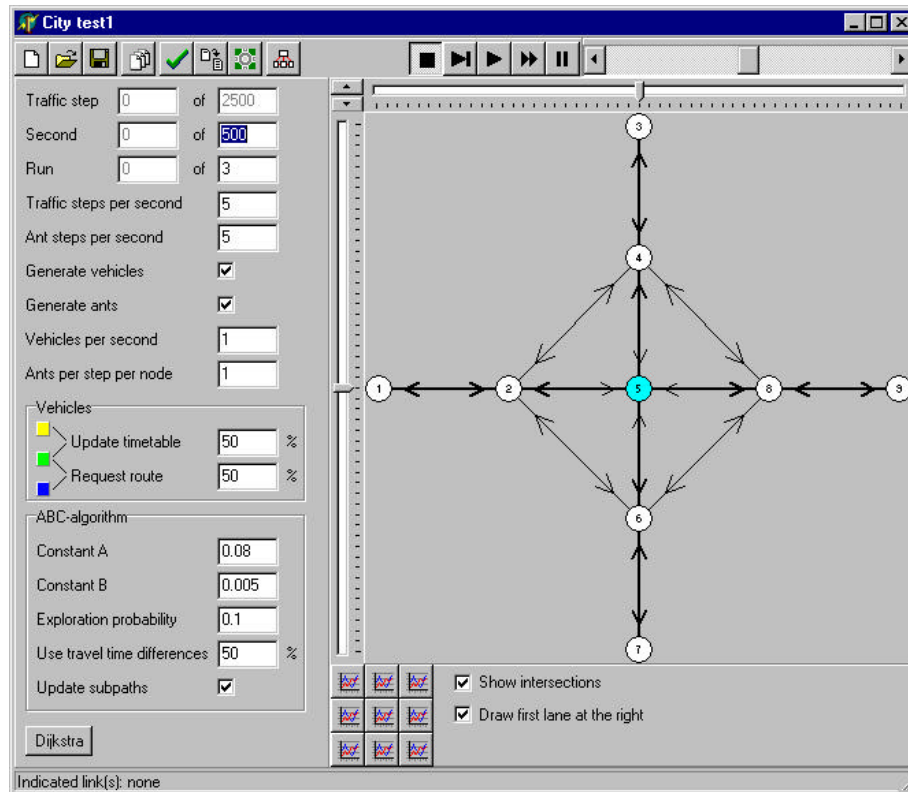


Figure 155: The City program has loaded a simulation environment

- Now use the -button to check whether the data are complete. If the program does not respond with the message 'Data complete', the data have to be completed first before running a simulation. The simplest way to accomplish this is by using the -button, which automatically supplements missing data with default values. These changes must be saved by using the -button. Check again if the data are complete. If the data are still not complete, they probably have to be adjusted manually by editing the data files. How this is done is explained in section 4.2.1.
- If there are any roundabouts in the traffic network, the -button has to be used to transform these intersections to real roundabouts. Roundabouts can be distinguished by light blue circles. If there is any doubt about whether there are roundabouts in the environment, just push the button. It will not have any effect if there are no roundabouts.
- Now push the -button to start the (distributed) Routing system. This may take some time for very large traffic networks. Wait until the program(s) appear(s) minimised in the task bar. If sound is enabled you will here a beep when ready.
- If wanted the settings at the left panel can be changed to the desired values but the defaults will work as well.
- Use the -button to start running the simulation.
- Upon closing the City program, do not forget to close the Routing system parts too. Do this by first clicking on the icons at the taskbar.

C.2 Features of the City program

Now we will explain the features of the program one by one. The order is the same as seen on the screen when the program is opened (see Figure 155). First we discuss the features of the City program and then the features of the Routing system program.

Top/left button group



Create empty city

This button will open a new empty environment. The current environment is closed and the view of the traffic network will be empty.



Open existing city

Use this button to open an existing environment. The current environment will be discarded.



Save this city

Save this environment with all data belonging to the traffic network. The parameter settings on the right panel will not be saved.



File names

A dialog will popup with all used filenames for this environment. These filenames can be changed then. Changes will affect the used files when saving the environment.



Check if data complete

Pushing this button will check for all intersections and all roads if they are complete. First all intersections are checked until the first error. When an error is found it is reported by displaying a message box. Further intersections will not be checked. Then all roads are checked until the first error. This error is also reported and no more roads are checked. If no errors are found, the program will respond with the message 'Data complete'.



Supplement missing data

This will supplement the missing data to the intersections and roads. Default values will be used. This should take care of most errors occurring when checking if the data is complete. However this will not solve the problem of errors in intersection and road numbers, neither will it solve the problem of missing incoming or outgoing roads at an intersection. These problems have to be worked out by editing the data files manually.



Transform roundabouts

This button will cause the transformation of all roundabout intersections in the environment. It will create new intersections connected in the shape of a circle and remove the original intersection(s).




Start distributed routing system

Pushing this button will start the (distributed) Routing system. This can be just one program or a group of programs, connected by communication via TCP/IP. This program will guide some of the vehicles in the traffic simulation. The City program will respond with a beep when the Routing system is ready.

Top/right button group




Stop and reset simulation

This button is down when the program is started. It indicates that the simulation is not running. Pushing this button will stop a running simulation and reset all data that is produced after starting a simulation, among which the graphs. When the simulation should be interrupted to check the current status and perhaps be continued afterwards, use the -button instead.




Do one step of the simulation

Pushing this button will execute one traffic step of the simulation. After this step the simulation will go into the paused state indicated by the -button. When the number of traffic steps per second is the same as the number of agent steps per second then the agents in the Routing system will also make one step. But when the number differs, the agents may execute more steps or no step at all.



Run simulation

Use this button to start running the simulation at normal speed. With a small environment this means that about one second of the simulation is executed in one real-time second. The larger the environment the slower the simulation runs. The speed can be adjusted with the scrollbar at the top/right. When the maximum number of seconds of a simulation has passed the next run (if any) is automatically started and the current second is set to zero. This means that all traffic is removed from the network and the process starts all over again. The newly acquired data is added to the old data already in the graphs. The data from the graphs are not cleared until the -button is pushed.



Run simulation at maximum speed (no visual change)

When using this button the simulation will run at maximum speed. To increase the speed the traffic is not moved visually. Updating the screen can sometimes be time-consuming, so skipping this might improve the maximum speed.




Pause simulation

Push this button when you want to interrupt the simulation. The simulation can be continued afterwards.



Simulation speed

Use this scrollbar to control the simulation speed. Move the bar to the left to slow down the speed and to the right to increase speed. When the bar is placed in the middle the speed of the simulation will be about real-time, assuming the environment is not too large. This scrollbar will only affect the speed when the -button is pushed.

Left panel with parameters

Figure 156: The parameters of the simulation

- *Traffic step*: the number of steps that the traffic in the simulation makes. The first box shows the number of steps already done. The second box shows the maximum number of steps per run of a simulation. This value is the multiplication of the parameters ‘Second’ and ‘Traffic steps per second’.
- *Second*: the number of seconds that a run of a simulation takes. The first box shows the number of seconds passed. The second box shows the maximum number of seconds per run of a simulation. This value can be adjusted in a range from 0 to 999,999.
- *Run*: the number of runs of a simulation. The simulation can be run a number of times without a pause. The results of different runs will be different, because a randomiser is used. The first box shows the current run. The second box shows the maximum number of runs. This value can be adjusted in a range from 0 to 999,999.
- *Traffic steps per second*: the number of steps that the vehicles will make per second. This value can range from 0 to 99. Setting this value to low will result in an inaccurate traffic simulation. Setting this value to high will make an unnecessarily high demand on the computer system.
- *Ant steps per second*: the number of steps that the ants will make per second. This value can range from 0 to 99. The ants will hop from one node to another every step. So this value determines the speed of the ants.
- *Generate vehicles*: whether the simulation should run with or without vehicles. In most cases it would be useless to run a simulation without vehicles.
- *Generate ants*: whether the Routing system should generate ants. Not generating ants would make it useless for vehicles to update the timetable or request a route from the Routing system (see concerning parameters), because the probability tables will not be updated.
- *Vehicles per second*: the number of new vehicles per second. At the beginning of a run of a simulation there is no traffic at all. This parameter controls the number of vehicles that will be added per second. The value can range from 0.00 to 9,999.

- *Ants per steps per node*: the number of new ants generated at every node and every step. The value can range from 0.00 to 9,999.
- Vehicles*:
- *Update timetable*: the percentage of the vehicles that will send update information to the timetable of the Routing system. This value can range from 0 to 100. These vehicles will be yellow or green: yellow if they do not also request a route from the Routing system, and green if they do.
 - *Request route*: the percentage of the vehicles that will request a route from the Routing system. This value can range from 0 to 100. These vehicles will be green or blue: green if they also send update information to the Routing system, and blue if they do not. The white vehicles in the simulation do not send update information to the Routing system nor do they request a route from the Routing system.
- ABC-algorithm*:
- *Constant A*: a parameter used in the algorithm to update the probability tables. The value can range from 0.0000 to 999,999. See the description of the ABC-algorithm for further details.
 - *Constant B*: a parameter used in the algorithm to update the probability tables. The value can range from 0.000 to 999,999. See the description of the ABC-algorithm for further details.
 - *Exploration probability*: a parameter used in the algorithm to update the probability tables. The value can range from 0.0000 to 999,999. See the description of the ABC-algorithm for further details.
 - *Use travel time differences*: instead of the real travel time of the vehicles the ABC-algorithm can also use the difference between the real travel time and the minimum travel time. Use this parameter to define how the time is composed from these two options. The value can range from 0 to 100.
 - *Update subpaths*: check this option when the ants should not only update the route to their destination, but also the route to all other intersections on their way.

The button at the bottom is not actually a parameter:

- *Dijkstra*: pushing this button will start the computation of the shortest paths with the aid of Dijkstra's algorithm. The algorithm uses the length and speed per road to compute the time per road. The outcome only applies to an environment where there is no delay. The results will be stored at the routing tables of the intersections in the traffic network. This way all vehicles that do not use the Routing system will follow the resulting routes of this computation.

Traffic network view

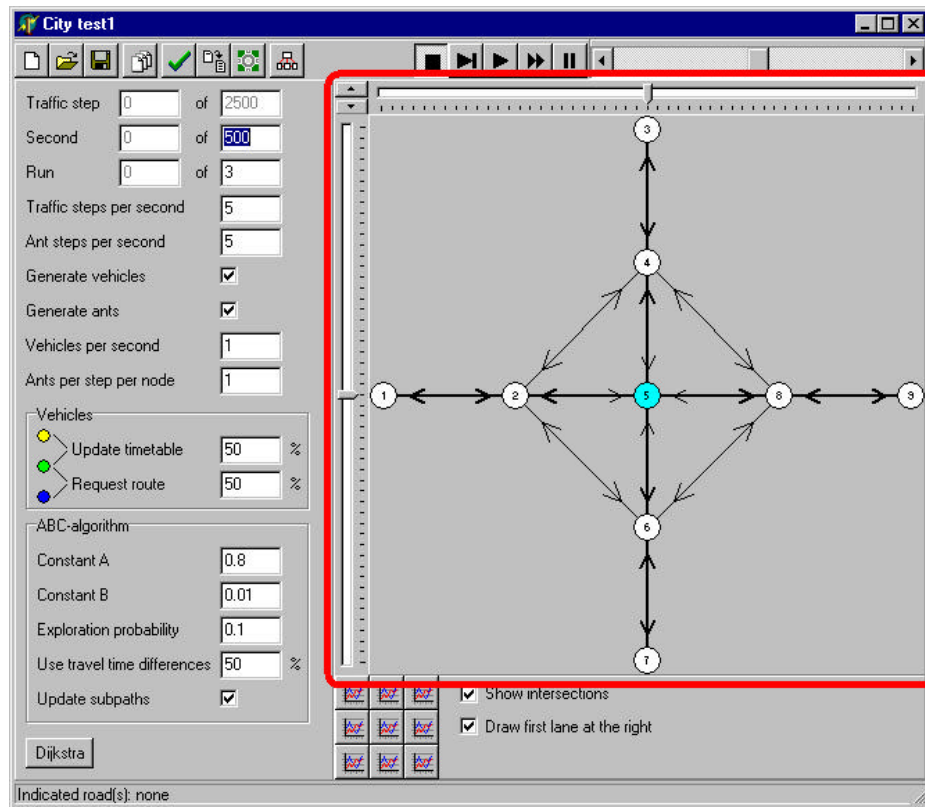


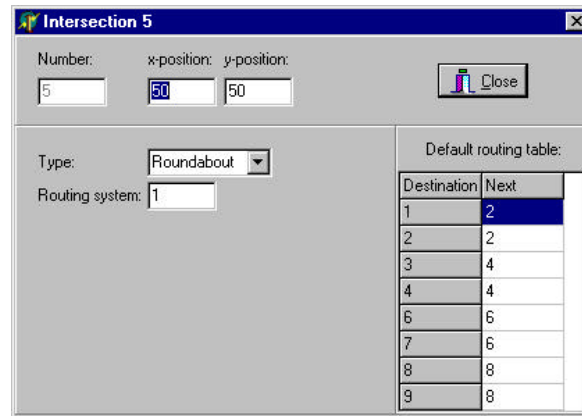
Figure 157: The traffic network view

- Zooming**

When a traffic environment is loaded the program will show it in the traffic network view. Upon loading the entire network will fit in the view, but it is possible to zoom in on a specific part of the network. The easiest way to do this is to point the pointers of the track bars to the location where you want to zoom into. The track bars are located at the top and to the left of the network view. Then use the up-arrow of the up-down control at the top left of the network view to zoom in. Use the down-arrow to zoom out again.

- *Intersections*

In the network view you can click on an intersection. This will popup an intersection view like in Figure 158.



Destination	Next
1	2
2	2
3	4
4	4
6	6
7	6
8	8
9	8

Figure 158: Intersection view

In this view there is a field with the number of the intersection. This field cannot be changed. Next to that the x- and y-coordinates can be changed. The type can be set to *Normal*, *Traffic lights* or *Roundabout*. The colour of the intersection depends on this type. Normal intersections are white, yellow is used for traffic lights, and the roundabouts can be distinguished by their blue colour. The number in the field of the Routing system determines which Routing system part takes care of this intersection. The Routing system can consist of different parts. How many parts, depends on how many different numbers there are for the Routing system. The number can range from 1 to 999. The default routing table contains two columns: one with the possible destinations and one with the next intersection to go to. The vehicles that do not use the Routing system to guide them will use this table to find the route. For example when a vehicle reaches intersection 5 and uses the default routing table shown in Figure 158 then it reads from the table that it has to go via intersection 4 when it wants to reach intersection 3. This table can be adjusted manually or it can be computed by using the button called 'Dijkstra' (reopen the view to see the changes). Push 'Close' to close the view. Changes are accomplished upon leaving the view.

- *Roads (left-clicking)*

When left-clicking on a road the following view will popup.

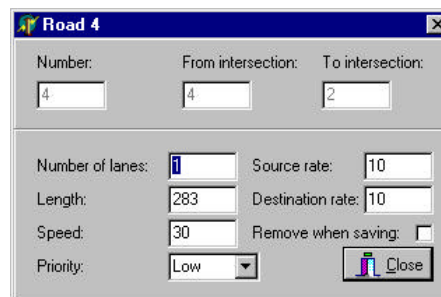



Figure 159: Road view

The first field in this view is the road number. Then there are two fields with intersection numbers. The first is where the road starts and the second is where the road ends. Note that the roads are directed, so the road with the numbers in reversed order is a separate road. These fields cannot be adjusted. One can change the number of lanes from 1 to 9. The length (in meters) can be set in the range from 1 to 9999. The speed can range from 1 to 999 and is measured in km/h. All

vehicles on a road will drive with this speed, unless of course they are hindered by an intersection or other waiting vehicles. The priority of a road can be changed to *High* or *Low*. This setting is used to determine whether a vehicle should wait for vehicles on other roads when crossing an intersection according to the precedence rules. The source rate can be changed from 0 to 99,999. The higher this number the more vehicles will (on average) start on this road. When it is zero there will not start any vehicles on this road. The same range from 0 to 99,999 goes for the destination rate but the higher this number the more vehicles will have this road as a destination. When you check the field 'Remove when saving' this road will not be saved when the -button is pushed. This will be noticed when reopening the environment. Push 'Close' to close the view. Changes are accomplished upon leaving the view.

- *Roads (right-clicking)*
When right-clicking on a road you will be asked to disable or enable the road. Disabled roads will be shown red instead of black.
- *Vehicles*
When there are vehicles in the network view you can also click on them. This will popup a message box with some properties of the vehicle. These properties are: the vehicle number, the source road, the current road and the destination road. And if the vehicle requests its route from the Routing system and it has any intersections left in its memory, it will show the next intersection of its route. This 'next' intersection is not the one it is heading for, but the one after that, because the intersection where it is heading for is trivial when looking at the movement of the vehicle. This information can be used to track a vehicle and foresee where it is going.

Bottom/right buttons and options

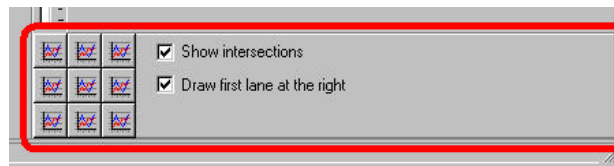



Figure 160: Graphs and network view options

At the bottom/right of the program there are nine buttons and two checkboxes. Behind every button is a different graph. Move the mouse cursor over a button to see what graph it will show. The graphs will be filled with data when running a simulation. They can be closed and (re)opened at any time. Any graph can be saved to an Enhanced Windows Metafile (.emf). To do this, use the -button on the graph view. Along the x-axis of all graphs is the number of time steps of a run. These steps are measured in traffic steps. This can be different from the number of ant steps. The available graphs are:

- *Total number of ants*
This graph shows the number of ants in (all parts of) the Routing system at every time step.
- *Total number of backward ants*
This graph shows the number of backward ants in the Routing system at every time step. This will be a fraction of the total number of ants, because all ants are forward ants or backward ants.
- *Average living ant age*
This graph shows the average age of the vehicles that are currently in the Routing system. Note that the average age at which the ants die is about twice this value. The age is measured in the number of steps that the ant has made.
- *Total number of differing next nodes*
The Routing system uses a probability table to route the vehicles. Because this table changes dynamically during a simulation it will be different from its initial state. In its initial state the intersections with the highest probabilities are the same as the intersections in the default routing tables. The intersections with the highest probabilities that differ from the initial state are counted and represented in this graph.
- *Average standard vehicle age*

This graph shows the average age of the standard vehicles, which are currently in the traffic network. Standard vehicles are the ones that do not use the Routing system to be guided. Note that the average age at which the vehicles reach their destination is about twice the value in the graph. The age is measured in seconds since the vehicle started.

- *Average smart vehicle age*

This graph shows the average age of the smart vehicles, which are currently in the traffic network. Smart vehicles are the ones that use the Routing system to find their route. Note that the average age at which the vehicles reach their destination is about twice the value in the graph. The age is measured in seconds since the vehicle started.

- *Total number of vehicles*

This graph shows the total number of vehicles currently in the traffic network.

- *Average standard route time*

This graph shows the average route times of all standard vehicles that ever arrived at their destination since the start of the run of the simulation. Standard vehicles are the ones that do not use the Routing system to be guided. The time is measured in seconds.

- *Average smart route time*


This graph shows the average route times of all smart vehicles that ever arrived at their destination since the start of the run of the simulation. Smart vehicles are the ones that use the Routing system to find their route. The time is measured in seconds.

The checkbox 'Show intersections' can be unchecked to hide the intersections. Especially in large networks this will make the roads much clearer. The hidden intersections cannot be clicked to open the intersection view. Vehicles are always drawn at the right side of the road. But when a road has two or more lanes, the vehicles have to be drawn next to each other (and at the right side of the road). Normally the vehicle in the first lane would be drawn most right and a vehicle in the last lane would be drawn most left, but directly right from the road. In this case when there is a vehicle in the first lane and none in any other lane there will be a gap between the vehicle and the road. This gap will only be filled when another vehicle comes next to the first in the second lane. Because vehicles will make use of the first lane as long as they can, this gap will appear rather often on roads with more than one lane. And especially when the traffic network is large it may be difficult to recognise on what road a vehicle is driving. Therefore you can uncheck the checkbox 'Draw first lane at the right'. In that case the vehicles on the first lane will be drawn directly at the right of the lane, without a gap. Note that the difference can only be seen on roads with more than one lane. These two checkboxes do not influence the running of the simulation at all. They only have visual effects.

Status bar at the bottom

At the bottom of the program there is a status bar. This status bar indicates which roads the mouse pointer is pointing at. The mouse pointer can point at several roads at the same time when these roads are close or on top of each other, because the mouse does not have to point exactly on the road. This may be useful to recognise roads without having to click on them to see the road number.

C.3 Features of the Routing system program

A Routing system program can only be started by using the -button from the City program. It will be started minimised, that is, the icon of the program is only visible in the taskbar at the bottom of the screen. Click on this icon to see the program. Note that there can be several Routing system parts opened for one simulation. The program will look something like the following picture (Figure 161).

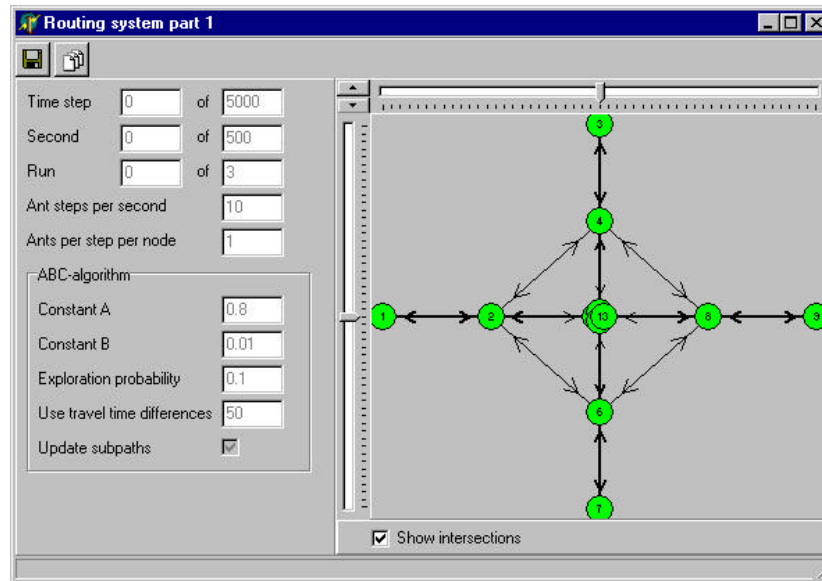


Figure 161: The Routing system program

The features of the Routing system program are discussed in the order in which they appear on the screen. So first the left side of the program is explained as indicated in Figure 162.

Left side of the Routing system program

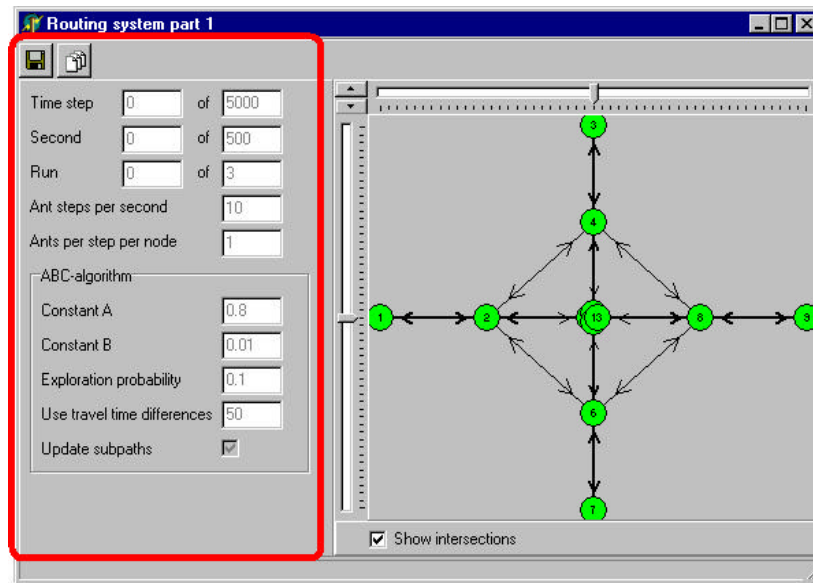


Figure 162: The left side of the Routing system program



Save the probability tables as default routing tables

This button will popup a dialog in which you can enter a name for the file to save. What is saved, are the probability tables in the form of a default routing table. So from the probability tables the highest value in each row is used. The concerning intersection is used to construct a default routing table, which can be used to route traffic that does not request the Routing system for a route. Only the intersections that are handled by the current part of the Routing system are save. In other words, only the green intersections will be saved in the file. This is because the probability tables from the intersections handled by other Routing system parts are unknown at this Routing system part.



File names

A dialog will popup with all used filenames from this environment that are necessary for the Routing system to work. These names cannot be changed here. They are copied from the City program.

The parameters of the Routing system cannot be changed in the Routing system program. They should be changed in the City program and these changes will then be sent to the Routing system parts.

- **Time step:** the number of steps that the ants in the Routing system make. The first box shows the number of steps already done. The second box shows the maximum number of steps per run of a simulation. This value is the multiplication of the parameters 'Second' and 'Ant steps per second'.
- **Second:** the number of seconds that a run of a simulation takes. The first box shows the number of seconds passed. The second box shows the maximum number of seconds per run of a simulation.
- **Run:** the number of runs of a simulation. The simulation can be run a number of times without a pause. The results of different runs will be different, because a randomiser is used. The first box shows the current run. The second box shows the maximum number of runs.
- **Ant steps per second:** the number of steps that the ants will make per second. The ants will hop from one node to another every step. So this value determines the speed of the ants.
- **Ants per step per node:** the number of new ants generated at every node and every step.

ABC-algorithm:

- *Constant A*: a parameter used in the algorithm to update the probability tables. See section 3.4.2 for a description of the ABC-algorithm.
- *Constant B*: a parameter used in the algorithm to update the probability tables. See section 3.4.2 for a description of the ABC-algorithm.
- *Exploration probability*: a parameter used in the algorithm to update the probability tables. See section 3.4.2 for a description of the ABC-algorithm.
- *Use travel time differences*: instead of the real travel time of the vehicles the ABC-algorithm can also use the differences between the real travel time and the minimum travel time. Use this parameter to define how the time is composed from these two options.
- *Update subpaths*: when this option is checked the ants should not only update the route to their destination, but also the route to all other intersection on their way.

Right side of the Routing system program

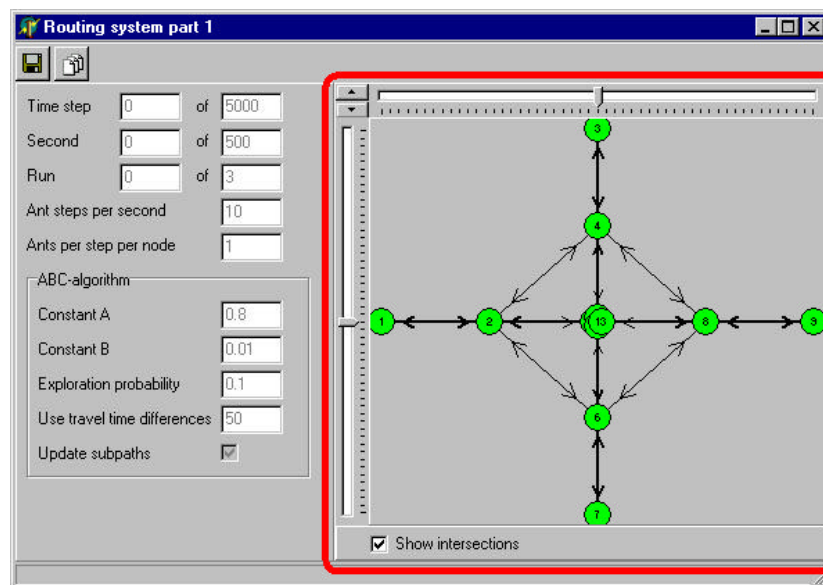
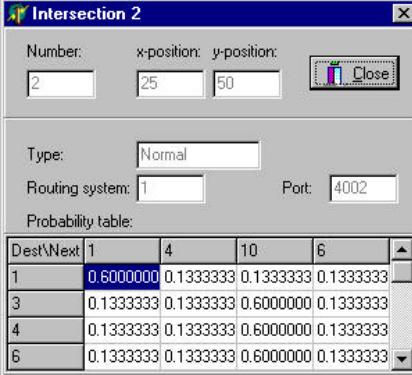


Figure 163: The right side of the Routing system program

- *Zooming*
The traffic network view will not only be visible in the City program, but also in any Routing system part. Here too you can zoom in on a specific part of the network. The easiest way to do this is to point the pointers of the track bars to the location where you want to zoom in to. The track bars are located at the top and to the left of the network view. Then use the up-arrow of the up-down control at the top left of the network view to zoom in. Use the down-arrow to zoom out again.

- *Intersections*

In the network view you can click on an intersection. This will popup an intersection view like in Figure 164.



Dest\Next	1	4	10	6
1	0.6000000	0.1333333	0.1333333	0.1333333
3	0.1333333	0.1333333	0.6000000	0.1333333
4	0.1333333	0.1333333	0.6000000	0.1333333
6	0.1333333	0.1333333	0.6000000	0.1333333

Figure 164: Intersection view

In this view none of the values can be changed. The first field shows the number of the intersection. Next to that are the x- and y-coordinates. The field type can be *Normal* or *Traffic lights*. Note that the type cannot be *Roundabout* because roundabouts are automatically transformed to a set of normal intersections when loaded in the Routing system. The number in the field of the Routing system determines which Routing system part takes care of this intersection. So if this number is equal to the number of this Routing system part then this intersection is handled by this Routing system part. In that case the intersection will be green. All other intersections that are handled by another part of the Routing system are white. The field for the port is the port number that TCP/IP uses for the communication between the different parts of the Routing system and the City program. In the probability table in the first column all possible destinations are enumerated. In the other columns there is a probability for every neighbouring node. All probabilities are set to zero if this intersection is not handled by this Routing system part (the intersection is white). Otherwise the sum of the probabilities per row is always one. These probabilities represent the chance for an ant to take the appropriate next intersection given their destination. The highest probability in a row is used for the vehicles. They will be sent to the next intersection with the highest probability considering their destination. Push 'Close' to close the view. To see the changes that the ants have made in the probability table, close the view and reopen it again.

- *Roads (left-clicking)*
When left-clicking on a road the following view will popup.

The screenshot shows a dialog box titled "Road 6" with a close button (X) in the top right corner. The dialog contains several input fields and a "Close" button. The fields are arranged in two rows. The first row contains "Number:" with a text box containing "6", "From intersection:" with a text box containing "10", and "To intersection:" with a text box containing "2". The second row contains "Number of lanes:" with a text box containing "1", "Length:" with a text box containing "200", "Speed:" with a text box containing "50", and "Priority:" with a text box containing "High". To the right of the "Length" and "Speed" fields is a note: "(right click on road for travel times)". At the bottom right is a "Close" button with a small icon.

Figure 165: Road view

None of the values in this view can be changed. The first field in this view is the road number. Then there are two fields with intersection numbers. The first is where the road starts and the second is where the road ends. Note that the roads are directed, so the road with the numbers in reversed order is a separate road. The field for the number of lanes is not used in the Routing system. The fields for length and speed are used to compute the travel time of a road when there is no congestion. The field for the priority is also useless in the Routing system. Push 'Close' to close the view.

- *Roads (right-clicking)*
When right-clicking on a road two message boxes will popup (per road). In the first message box all the measurements stored for this link are shown together with the time at which they were received. These measurements are the computed travel times from the vehicles that send updates. These measurements are used to compute some weighted average for the travel time of the road. This result is shown in the second message box. This travel time is used as a virtual delay for the ants.

The checkbox 'Show intersections' can be unchecked to hide the intersections. Especially in large networks this will make the roads much clearer. The hidden intersections cannot be clicked to open the intersection view.