Applying delegate multi-agent systems in a traffic control system

Elise Huard
Dirk Gorissen
Tom Holvoet

Report CW467, June 2006

Katholieke Universiteit Leuven
Department of Computer Science
Celestijnenlaan 200A – B-3001 Heverlee (Belgium)

Applying delegate multi-agent systems in a traffic control system

Elise Huard
Dirk Gorissen
Tom Holvoet

Report CW 467, June 2006

Department of Computer Science, K.U.Leuven

Abstract

Multi-Agent Systems have been proven to be a useful paradigm for solving complex coordination and control problems involving large numbers of autonomous entities interacting in a dynamic environment. Traffic Control is one such problem. In this report we present a novel software architecture based on "Delegate-MAS" that implements simplified traffic control in a pro-active manner by trying to predict and avoid road congestion. The report describes the development process and presents a proof-of-principle implementation to validate the approach.

AMS(MOS) Classification : Primary : D.2.

Contents

1.1 Traffic Control System

This report describes the design of a multiagent system (MAS) software architecture for a simplified Traffic Control System (TCS). A TCS systems tries to optimize the use of the road network in a proactive way by trying to predict and avoid road congestion. Traffic Control (like manufacturing contol and web service coordination) is an example of a *coordination and control*-class application and it has been argued [6] that the MAS paradigm is a suitable approach to modeling these kinds of applications. Characteristics of such applications include:

- There is an underlying (physical or software) system that needs to be controlled by a top layer software system.
- The top layer works at a much faster speed, it can plan ahead.
- It is a task-oriented application domain where a task entails moving through the environment (mobile entities) and performing operations using resources (static entities).
- Constitutes of a large number of distributed entities in a very dynamic environment.
- Complex functional / non-functional requirements.
- Centralized software architectures are typically unsuitable.

In agent based TC agents (software or robotic) are located in different locations, receive sensor data that are geographically distributed, and must coordinate their actions in order to ensure global system optimality [8]. Research into using MAS for solving TC is not new and has been explored by various researchers [5, 11] and private companies [2, 3].

In what follows we present the different stages of the development process (conform the principles of the Universal Process (UP)) en route to our software architecture. A simple proof-of-principle implementation will also be provided to validate the proposed design. The report concludes with a critical evaluation and pointers to future improvements.

1.2 Development Process

1.2.1 Practical remarks

The development of this project was carried out with the Eclipse platform [1]. Matlab was used for experimentation and testing. The integral implementation, including the test scenarios, is available at:
http://www.cs.kuleuven.be/~danny/DelegateMAS/Traffic-Control-DelegateMAS.zip

1.2.2 Domain Model

The domain model is that of a simplified TCS that attempts to deal with congestion in a pro-active manner. Such a system supports the coordinated navigation of many cars in a road network, avoiding as much as possible that congestion arises in the (near) future, and thus achieving a more efficient use of the road network. The system relies on smart cars as well as smart roads which are equipped with electronic devices, sensors and communication hardware. Smart cars support the driver by providing up-to-date information or even partially or fully automated driving itself. Smart roads monitor traffic and support the smart cars in their decision making process. For a list of the exact simplifications made see section 1.2.3.2.

1.2.3 Requirements Analysis

The requirements of the project have been established using, where applicable, the FURPS+ model.

1.2.3.1 Functional requirements

These specify everything that a user of the system would need to know regarding what the system does.

1. The TCS should support the coordinated navigation of many cars in a road network, trying to avoid congestion in the (near) future as much possible, and thus achieve a more efficient use of the road infrastructure. Given a road network with a number of bottlenecks the application of the implemented pro-active TCS should result in a significant decrease in congestion over more simpler traffic control methods (ie. all drivers take the shortest path to their destination).
2. The performance of the TCS must be measurable. There must be ways to establish: the time cars have spent standing still due to congestion and the time (or cycles) they took reaching their destination (routing efficiency).

1.2.3.2 Non-functional requirements

The non-functional requirements describe constraints on the operation or development of the system [4]. We also include the domain simplifications made as operation constraints.

Implementation constraints:

- Adopt a MAS problem decomposition.
- Implement using an Object-Oriented (OO) programming language (C++, C# or Java)

Problem simplifications:

- Synchronous computation and actions for all components of the system (ie. all cars move in lock-step)
- All roads have a length and a direction
- A road is discretized into segments of fixed length.
- Crossroads have incoming and outgoing roads
- At any timestep, a road segment can be occupied by at most one car.
- All cars move at the same speed: 0 or 1 segments per time step
- All cars and sections of roads are equipped with electronic sensors and communication capabilities. One road segment is able to communicate with all segments within a user specified range and with the car occupying it. Likewise a car is only able to communicate with the road segment is is currently situated on.
- Each car maintains simple state information: current position and trip information (start location and destination)

- Each car is behaves completely deterministically, it adheres fully to the suggestions made by the TCS.

Evolution points (possible extensions) :

- Dynamic road network (allowance for road works, obstruction, new roads ...)
- Non-deterministic car behavior (include the possibility that the driver doesn't always follow the system's advice)
- Asynchronous computation and actions
- Cars have different speeds.

1.2.4 Mapping Requirements to Architecture

In this section we consider how the requirements mentioned above may be mapped onto a concrete MAS based software architecture.

1.2.4.1 Describing factors

The requirements constitute factors which have an impact on the architecture. The following table describes the requirements and their consequences ([7]).

Factor	**Measures and quality scenarios**	**Impact of factor on architecture**	**Priority for success**	**Difficulty or risk**
proactive traffic congestion system	A significant difference in the time spent in congestion compared to a scenario without traffic control.	A proactive congestion control system is not viable with a centralized system: a distributed system is required, if only for scalability.	H	H
measurable performance	Aat the end of every scenario the performance measures should be easily retrievable (time spent in congestion, distance travelled and total travelling time).	Use of standard output to display performance characteristics. Possibly making use of a logging library. This allows performance output without affecting the main functions	H	L

1.2.4.2 Resolution of architectural factors : major decisions

Now that the requirements and their impact have been detailed, the architecture can be designed as a solution to these requirements.

- ISSUE
  - Functionality : proactive traffic control system
- FACTOR

  - Significantly better than no traffic control system: improvement in travel time and decrease in time standing still (the two are correlated).

- SOLUTION

For a proactive traffic control system, centralized control is not realistic. The system must be scalable to an entire wide area road network. If centralized, this would require an enormous amount of communication, not to mention vast computing resources. A distributed system involving delegation is necessary.

A simplistic approach to decentralization would be to distribute the centralized system over a limited number of geographical regions in a hierarchical way. Such an approach has been adopted in other work [11], but rather as a way to deliver information to the drivers about meteorological conditions, preferred trajectories etc. So it was not used for the explicit routing of every car individually, this would demand a lot of planning and communication between nodes ('traffic managers' in [11]).

A fully decentralized solution involving autonomous agents for traffic routing, on the other hand, is very scalable. For this to work rules need to be defined at the agent level to ensure agents cooperate and the road network as a whole is used in an optimal way. Such decentralization also eases the handling of a dynamic road network.

1.2.4.3 Chosen Software Architecture

In this project we have chosen to adopt the Delegate-MAS software architecture described in [6]. This architecture (in the context of TC) consists of the following entities:

- **The Environment** is a dynamic directed graph through which agents move.
- **Task Agents** that try to accomplish a task by moving through the underlying resources (represented by resource agents). Eg: a car that needs to move from start to finish through the road network. In order to accomplish their goal task agents iterate through a BDI-like control loop. The difference with standard BDI is that the implementation of each step (collect beliefs, reason, select intention) is delegated to simple delegate agents called ants. These are simple reactive agents that traverse the environment using stigmergy to communicate and gather information.
- **Resource Agents** that manage the use of a certain resource. In this case the resource is a road segment, that can only be occupied by one car at a time. The task agents use resource agents to book a certain resource in advance, taking into account if the resource is free at that moment.
- **Delegate Agents** (ants):
  - simple, reactive agents that are created, sent out, and collected by task- and resource agents
  - virtual entities, not directly connected with anything physical
  - behaviour inspired by food foraging in ant colonies
  - communicate with other ants through the environment (stigmergy)
  - navigate virtually through the resources at a much faster rate than the task agent
  - 3 types[1]

[1]The feasibility and exploration ants serve similar purposes as the Forward and Backward ants in the AntNet system [10].

- *Feasibility Ants*: sent out by resource agents, gather information about the underlying environment itself (which roads lead to which destinations). These are the ants that detect changes to the environment (severed roads, new crossroads, ...)
- *Exploration Ants*: sent out by task agents, scouts that travel through the network gathering information about possible routes (how costly is a route)
- *Intention Ants*: sent out by task agents, fix an intention in the environment (reserve the best route). Task agents must create intention ants to refresh their intention at a frequency that is sufficiently high to maintain their reservations.

Other papers on the subject mention different techniques for implementing a TCS. One system doesn't consider road segments but only intersections, their management, and constraints on those intersections [5]. The agents are the individual cars and the intersections. The intersections, overpasses and crossroads keep track of reservations. The car calls ahead to the intersection to book its passage. The intersection is itself a NxN grid, and a car will occupy only a part of this grid. It is a BDI agent system, which uses planning, and messages - the car plots its course, and variable speeds are used to avoid congestion.

Other researchers study traffic control at a higher level. One of these has traffic manager agents, which communicate with meteo agents, plan agents and other information providing agents to manage the traffic flow [11] (communication using FIPA standards).

The reason for choosing a Delegate-MAS architecture instead of the standard BDI based architecture is because of the complexity and dynamics of the system we are trying to model (Traffic Control)[2]. TC is an example of a coordination and control problem and it has convincingly been argued that a Delegate-MAS architecture provides a suitable solution for such problems [6]. The disadvantages of a pure BDI approach include:

- How to represent knowledge and keep it up to date?
- Explicitly employing means-ends reasoning (planning) is far to costly and difficult for traffic control.
- The environment may change while the agents is deliberating, this means that chosen intentions are not guaranteed to stay optimal within one iteration.

A pure subsumption architecture would be also possible for traffic control. However, by definition such a system is not proactive, agents only act locally and on very short term, there is no learning. In addition designing subsumption rules for a complex problem as TC is no trivial undertaking.

A similar thing is true with purely reactive agents based solely on stigmergy (Ants). Though they have been used successfully in network traffic routing (AntNet [10]) in our case we would like agents with more intelligence and a stronger notion of state/objectives/.... This is especially true if our system were to be extended to a more realistic TCS.

Another possible MAS model is Maes' Behavior Network for Situated Agents. However, it too is not really suited for the TC problem (how to specify the behavioral links, how to keep the behavior graph up to date in an efficient way, ...). Maes' model is more suited to problems were accomplishing a goal can easily be stated in terms of accomplishing subgoals with inter dependencies.

A hybrid agent architecture (a combination between reactive and reasoning agents) them seems like a good approach that combines both the practical and reactive agent types. However, while this

[2]This is of course even more so for the Deductive Agent Architecture.

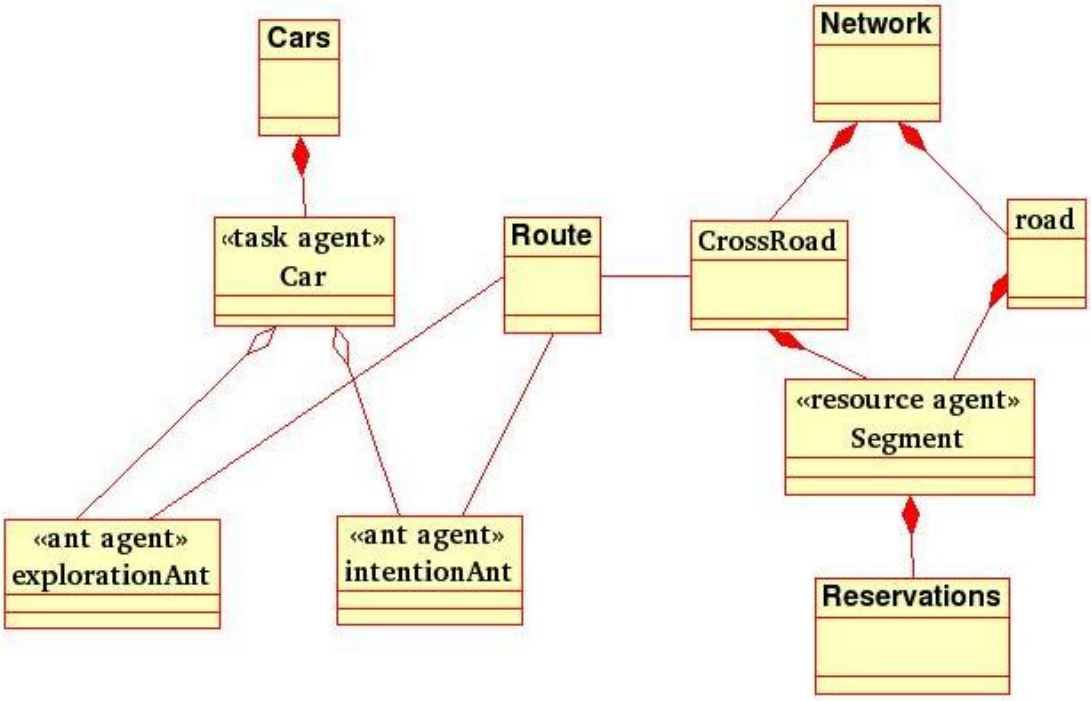

Figure 1.1: High Level Architecture Diagram

approach could be applied it is still more complex than the Delegate MAS architecture. Hybrid agents make use of a layered architecture and thus requires the designer to explictly define how these layers may interact without losing flexibility or fault tolerance. In contrast, the Delegate MAS approach gives us the flexibility and ease of implementation of reactive agents while still providing simple BDI-like abstractions that ease high level coordination and behavior control.

1.2.5 Concretizing the Software Architecture

In this section we now zoom in and describe the architecture in more detail.

To design the TCS we used a hybrid bottom-up / top-down approach. The top-down approach aids in understanding the control flow, the problem from a use-case point of view, while the bottom-up view helps to understand the responsibilities of each individual entity involved. Switching between both iteratively manner helped to better understand the problem domain and how to mold the solution in software.

The result of this process is depicted in figure 1.1.

1.2.5.1 Bottom-up

To get an overview of the different entities involved in the TCS we listed them in a CRC-card like manner.

- Car
  - Characterized by: location, position, an origin, a destination

  - Behavior: moves from origin to destination at a minimum cost (distance and/or time)

- Road
  - Characterized by: length (number of segments), direction, neighboring crossroads
  - Behavior: divided into segments, accommodates cars moving in the direction of the road between two neighboring crossroads

- Segment
  - Characterized by: keeps a reservation list holding a record of cars that have reserved this segment at a particular point in time
  - Behavior: may only contain one car at a time, reservations evaporate over time

- Crossroad
  - Characterized by: incoming and outgoing roads
  - Behavior: connect multiple roads, consist of one segment, potential traffic bottlenecks

- Traffic network
  - Characterized by: a list of roads and crossroads
  - Behavior: map shared by all cars

Now that we have a high level idea of how our system is composed we can consider how they interact.

1.2.5.2 Top-down

For the top-down part we used pseudo code to understand and define the control flow of the simplified TCS using a Delegate-MAS based design. Again note, of course, that we are dealing with a simplified, synchronized TCS which allows us to sidestep many potential issues that arise in a real, concurrent system. The control flow of our TCS is shown below:

```
Network TN; //The traffic infrastructure
Cars cars;  //List of cars

//Main loop
while(cars.notReachedDestination()) do
   //Let each car figure out the best route it can take.
   foreach(c : Cars)
      //Gather information from the environment, like
      //what are the possible routes from this location (Beliefs).
      c.think();

      //Explore the different options taking into account
      //congestion in a pro-active way (Desires).
      c.plan();
```

```
        //Decide what to do, reserve a route (Intentions).
        //This includes potentially revising the current intention.
        c.decide();
     end
     //Let each car move one step on its current route.
     cars.move();

     //In the spirit of Delegate-MAS we need some kind
     //of evaporation of information in the environment.
     //This is realized by evaporating route reservations.
     TN.decayReservations(eps);
end
```

Since we have opted for a Delegate-MAS based architecture the high level control flow is distinctly BDI in nature. It is in the steps `plan()` and `decide()` that the delegate ant-agents come into play. Note that, due to complexity, we do not consider feasibility ants (though this functionality could be easily added at a later stage). Also note that, while the final version of our TCS will use delegate agents the first iteration will not include this functionality. In the initial version the methods `think()`, `plan()`, and `decide()` will simply always choose the shortest path to the destination. This allows us to have a simple prototype up and running quickly and allows us to identify any design flaws early on. For the delegate versions the working is as follows:

```
think():
  //To keep things simple we do not use feasibility ants here,
  //we assume there is some navigational tool available which
  //is able to return different possible routes.
  routes = gatherPossibleRoutes(currentLoc, dest);

plan():
  //Explore the set of possible routes
  foreach(route : routes)
     ants[i] = new ExplorationAnt(route);
     ants[i].explore()
  end

  //Each ant keeps a record of the cost of each route.
  //The cost is defined as: cost = w_d*distance + w_w*timeWaited
  bestRoute = minimalCost(ants);

decide():
  //If the the new best route is better than the current
  //best route by 'delta' update our intention.
  if(reviseIntention(bestRoute,curRoute,delta))
     curRoute = bestRoute;
  end
```

```
//Fix our interest in the route in the environment
//by reserving the corresponding road segments.
ant = new IntentionAnt(curRoute);
ant.bookRoute();
```

From the pseudo-code we see our TCS will end up with four tunable parameters:

1. $w_{distance}, w_{waiting}$: the relative importance of the distance traveled versus the time spent waiting due to congestion. A value of $w_{waiting} = 0$ reduces our TCS to one without congestion control (simple shortest path). For a realistic situation $w_{distance}, w_{waiting}$ should probably be set equal to one another, since the aim of a traffic control system is mostly to optimize the time spent in traffic for every car.

2. δ: how quickly should we revise our intention : this will determine whether the car agents are cautious or bold. The task agents have an open-minded form of commitment, which means they are allowed to change their intentions.

3. ϵ: how quickly should reservations decay, the larger ϵ the longer they remain valid.

The actual values of these parameters can be tuned according to the requirements of the final user of the system.

Now that we have a good understanding (bottom-up and top-down) of how our TCS will work we are ready to start on the detailed design.

1.2.6 Detailed Design

In this section we condense the discussion above into the single, formalized class diagram shown in figure 1.2.

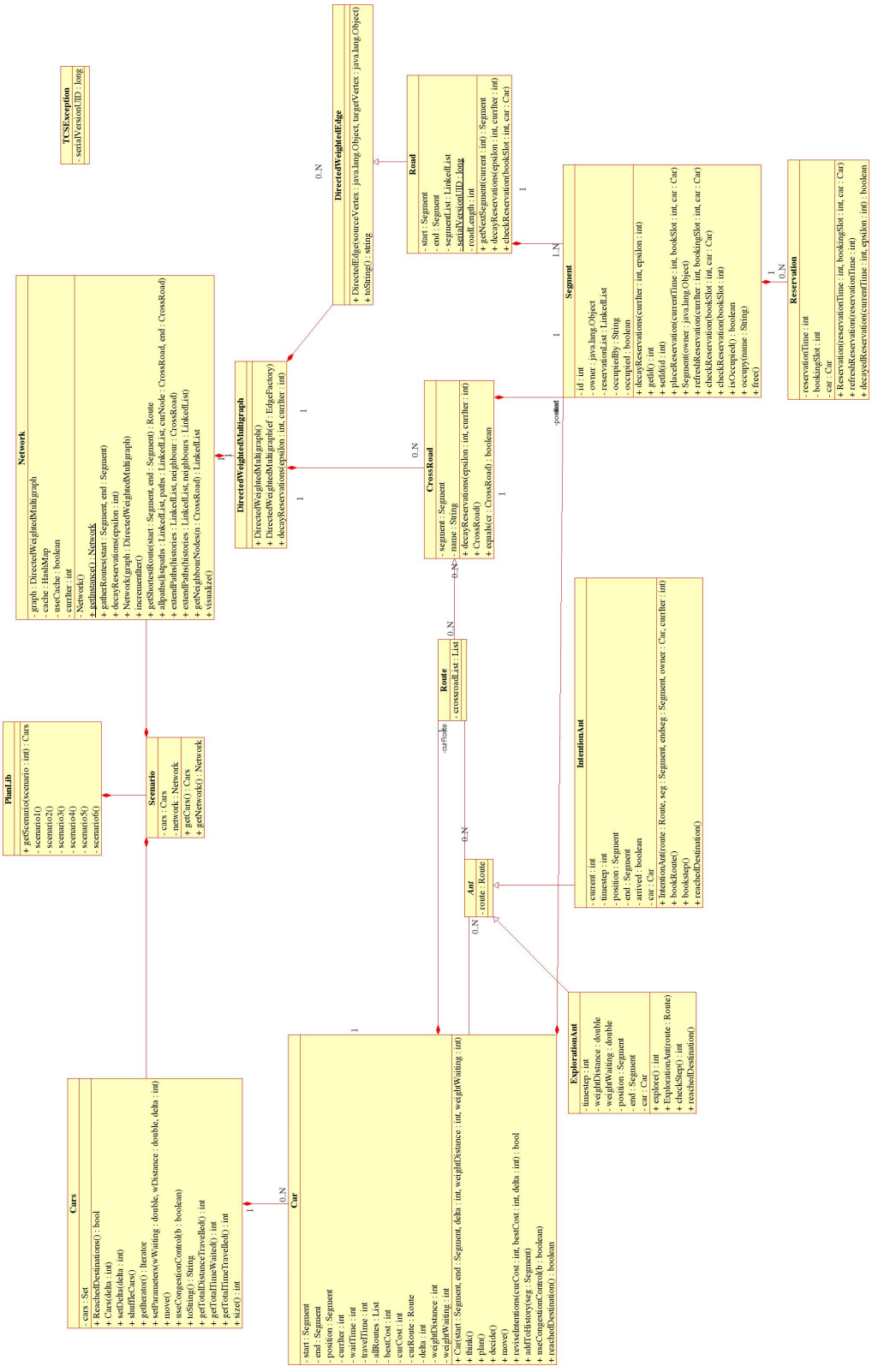

Figure 1.2: Class diagram

As can be seen from the diagram the traffic network is built upon an existing graph library (JGraphT). Otherwise the diagram should be self explanatory. Accessor methods were not included, their presence is implicit. This is the updated diagram, including the few methods that were added during development.

As mentioned in section 1.2.3.2 a sequential design was used in the scope of this project, to avoid the issues associated with multi-threaded designs (synchronization, mutual use of shared resources, deadlock). Multi-agent systems are parallel processes by nature, the activities of an agent should in theory be independent of what other agents do (concurrent problem solving), when they don't collaborate or use the same resources. However, multi-threaded design brings a lot of difficulties [9], so in the context of this project it was suggested we loop over all the agents in a sequential manner.

1.2.7 Implementation

The implementation of the TCS was done in two steps: first without pro-active congestion control (ie. simple shortest path routing) and then with it included. This allowed us to catch any design flaws before implementing the congestion control. However, it turned out that the implementation went very smoothly, no design related errors or problems were encountered. The implementation was done in Java 1.5.

1.3 Evaluation

In this section we run our implementation on two scenarios and investigate the impact adding congestion control has. One simple and one complex, lifelike scenario was considered. The scenarios are explained below together with the performance results.

1.3.1 Simple Scenario

The first scenario we consider is shown in figure 1.3. The cars are distributed as follows:

- 10 cars going from A to E
- 10 cars going from B to E

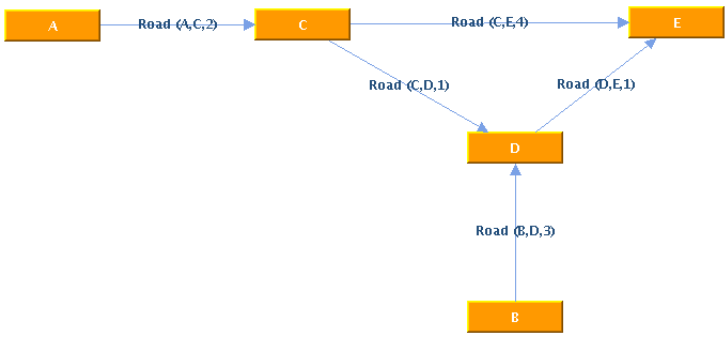

Figure 1.3: Map of the simple scenario

Studying the map one would expect congestion to arise at B if every car tries to follow the shortest route. If congestion control is switched on however, cars coming from A will notice a bottleneck at B

and take the longer route A-C-E instead. That this is indeed the case is shown by the program trace below (cars were randomly ordered on start).

```
-- Simulation Result : no congestion control ---
Car c1, path=A->E, history=[A, C, D, E], distance=7, timeTravelled=16, timeWaited=9
Car c5, path=A->E, history=[A, C, D, E], distance=7, timeTravelled=7, timeWaited=0
Car c3, path=A->E, history=[A, C, D, E], distance=7, timeTravelled=31, timeWaited=24
Car c1, path=A->E, history=[A, C, D, E], distance=7, timeTravelled=23, timeWaited=16
Car c6, path=B->E, history=[B, D, E], distance=5, timeTravelled=8, timeWaited=3
Car c2, path=B->E, history=[B, D, E], distance=5, timeTravelled=28, timeWaited=23
Car c1, path=A->E, history=[A, C, D, E], distance=7, timeTravelled=37, timeWaited=30
Car c4, path=B->E, history=[B, D, E], distance=5, timeTravelled=30, timeWaited=25
Car c7, path=A->E, history=[A, C, D, E], distance=7, timeTravelled=35, timeWaited=28
Car c10, path=B->E, history=[B, D, E], distance=5, timeTravelled=25, timeWaited=20
Car c9, path=A->E, history=[A, C, D, E], distance=7, timeTravelled=10, timeWaited=3
Car c5, path=A->E, history=[A, C, D, E], distance=7, timeTravelled=26, timeWaited=19
Car c2, path=B->E, history=[B, D, E], distance=5, timeTravelled=21, timeWaited=16
Car c8, path=B->E, history=[B, D, E], distance=5, timeTravelled=18, timeWaited=13
Car c3, path=A->E, history=[A, C, D, E], distance=7, timeTravelled=14, timeWaited=7
Car c3, path=A->E, history=[A, C, D, E], distance=7, timeTravelled=33, timeWaited=26
Car c5, path=A->E, history=[A, C, D, E], distance=7, timeTravelled=39, timeWaited=32
Car c2, path=B->E, history=[B, D, E], distance=5, timeTravelled=12, timeWaited=7
Car c4, path=B->E, history=[B, D, E], distance=5, timeTravelled=5, timeWaited=0
Car c4, path=B->E, history=[B, D, E], distance=5, timeTravelled=19, timeWaited=14

Total time spent in congestion: 315
Total distance travelled: 122
Total travel time: 437

-- Simulation Result : with congestion control---
Car c1, path=A->E, history=[A, C, D, E], distance=7, timeTravelled=24, timeWaited=17
Car c7, path=A->E, history=[A, C, E], distance=9, timeTravelled=26, timeWaited=17
Car c5, path=A->E, history=[A, C, D, E], distance=7, timeTravelled=7, timeWaited=0
Car c1, path=A->E, history=[A, C, D, E], distance=7, timeTravelled=26, timeWaited=19
Car c10, path=B->E, history=[B, D, E], distance=5, timeTravelled=29, timeWaited=24
Car c2, path=B->E, history=[B, D, E], distance=5, timeTravelled=13, timeWaited=8
Car c2, path=B->E, history=[B, D, E], distance=5, timeTravelled=5, timeWaited=0
Car c3, path=A->E, history=[A, C, E], distance=9, timeTravelled=22, timeWaited=13
Car c4, path=B->E, history=[B, D, E], distance=5, timeTravelled=22, timeWaited=17
Car c4, path=B->E, history=[B, D, E], distance=5, timeTravelled=18, timeWaited=13
Car c2, path=B->E, history=[B, D, E], distance=5, timeTravelled=14, timeWaited=9
Car c1, path=A->E, history=[A, C, E], distance=9, timeTravelled=12, timeWaited=3
Car c5, path=A->E, history=[A, C, E], distance=9, timeTravelled=14, timeWaited=5
Car c9, path=A->E, history=[A, C, D, E], distance=7, timeTravelled=11, timeWaited=4
Car c5, path=A->E, history=[A, C, D, E], distance=7, timeTravelled=16, timeWaited=9
Car c6, path=B->E, history=[B, D, E], distance=5, timeTravelled=9, timeWaited=4
Car c3, path=A->E, history=[A, C, E], distance=9, timeTravelled=20, timeWaited=11
Car c8, path=B->E, history=[B, D, E], distance=5, timeTravelled=28, timeWaited=23
Car c4, path=B->E, history=[B, D, E], distance=5, timeTravelled=20, timeWaited=15
Car c3, path=A->E, history=[A, C, E], distance=9, timeTravelled=18, timeWaited=9

Total time spent in congestion: 220
Total distance travelled: 134
Total travel time: 354
```

Given these figures we can now calculate the percentual difference in performance for this scenario. This is shown in table 1.1.

1.3.2 Complex Scenario

The second scenario is a more complex one and was inspired by a real life congestion point: the Leonard crossroad, which is the crossing between the Brussels Ring and the E411 motorway. This is a well-known point of congestion. The people commuting from Overijse or farther south into Brussels

	Timesteps Congested	Distance Travelled	Total Travel Time
Shortest Path	315	122	437
Pro-Active Congestion Control	220	134	354
% Difference	**-30%**	**+9%**	**-18%**

Table 1.1: Influence of Congestion Control ($w_{waiting} = 1, w_{distance} = 1, \delta = 1, \epsilon = 1$)

every morning have several alternatives to get into Brussels, and a few of them were taken into account in this scenario. This scenario was used to check how the TCS system would perform in a complex and demanding scenario, and whether it's really as scalable as it's supposed to be.

Figure 1.4 shows the map of this situation (alternative roads shown in colored lines). The corresponding graph is depicted in figure 1.5. Some extra traffic is added on the ring and on the E411, to add to the realism of the simulation. Only the morning traffic is considered, so cars are travelling from Overijse to Brussels, and in both directions on the ring.

The traffic streams are as follows (the numbers are obviously smaller than in the real scenario, to keep processing times reasonable):

- cars c1 to c100 go from A (Overijse) to G (Brussels)
- cars d1 to d50 go from F (south part of Ring-O) to I (north part of Ring-O)
- cars e1 to e50 go from I (north part of Ring-O) to F (south part of Ring-O)

This has the effect that the cars are leaked into the scenario at the edges, while in reality cars are liable to come from different points on the road. Since a large part of the scenario is motorway, however, this is likely to be close to reality. To make it more realistic, E411 has three lanes, and the ring has 2 lanes in both directions. The lanes are modeled as edges of the graph, with the possibility to change lanes in certain points (small symbolic edges).

Figure 1.4: The shortest path is indicated in red. Possible alternative routes are highlighted in other colors (yellow, green, blue).

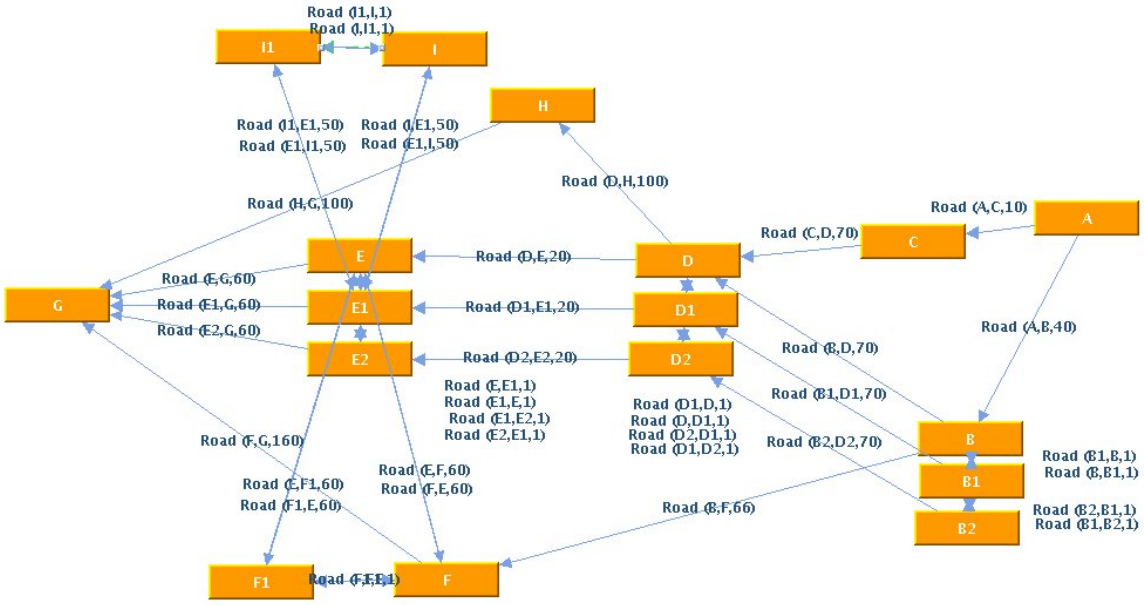

Figure 1.5: Graph model for the real scenario. 3 lanes on the E411, 2x2 lanes on the ring.

The corresponding performance figures of running the scenario twice are shown in table 1.2. The trace of the program is a bit too long to display here.

	Timesteps Congested	**Distance Travelled**	**Total Travel Time**
Shortest Path	14650	27800	42450
Pro-Active Congestion Control	10835	29050	39885
% Difference	**-26%**	**+4%**	**-6%**

Table 1.2: Influence of Congestion Control $(w_{waiting} = 1, w_{distance} = 1, \delta = 1, \epsilon = 1)$

The improvement was smaller than in the simple scenario in relative terms, but in absolute terms a lot was gained.

1.3.3 Parameter Tuning

We can now study the effect the tuning parameters have on the simulation. For this we shall use a simplified version of the complex scenario described above. The network graph is shown in figure 1.6. The traffic distribution is as follows:

- 50 cars going from A to G
- 25 cars going from I to F
- 25 cars going from F to I

Thus crossroad E is the bottleneck that must be avoided.

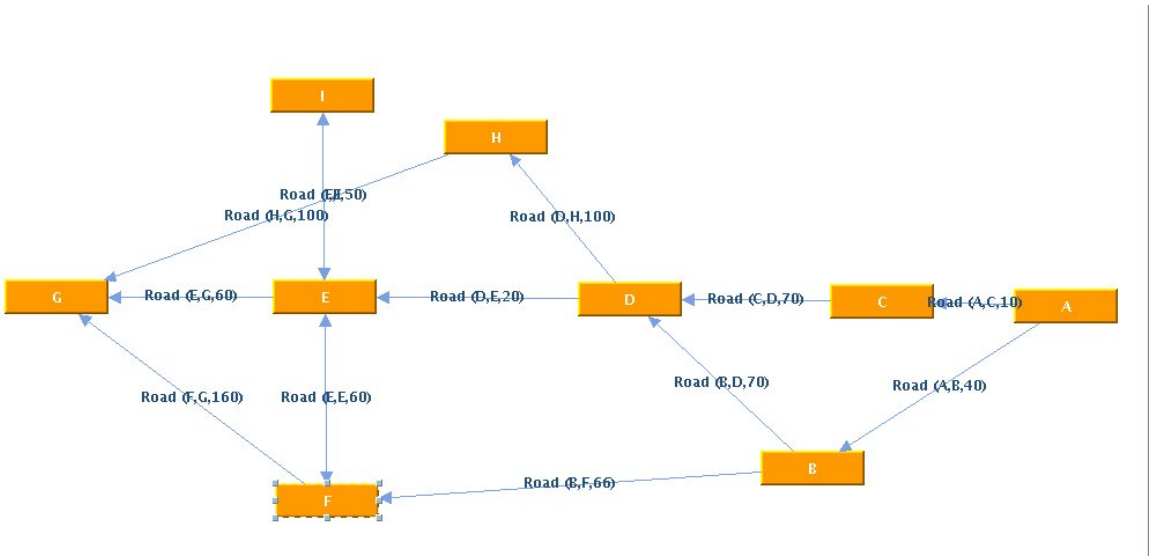

Figure 1.6: Model of the real scenario

Let us first consider the cost weights $w_{waiting}, w_{distance}$. Figures 1.7, 1.8, 1.9 show plots of the performance landscape for these parameters (using the complex scenario). The results are aggregated over all cars.

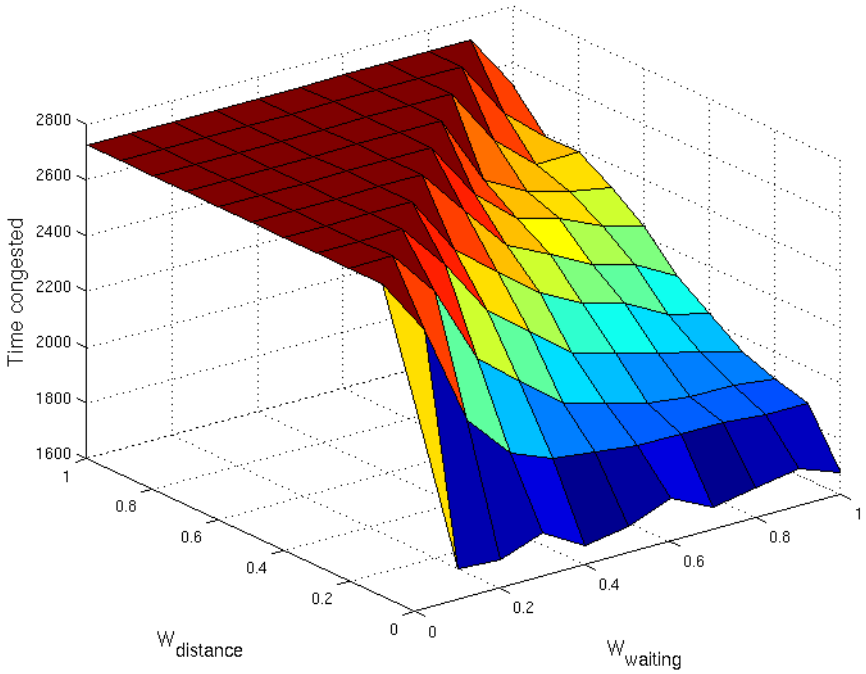

Figure 1.7: Performance landscape for $w_{waiting}, w_{distance}$ (timesteps congested)

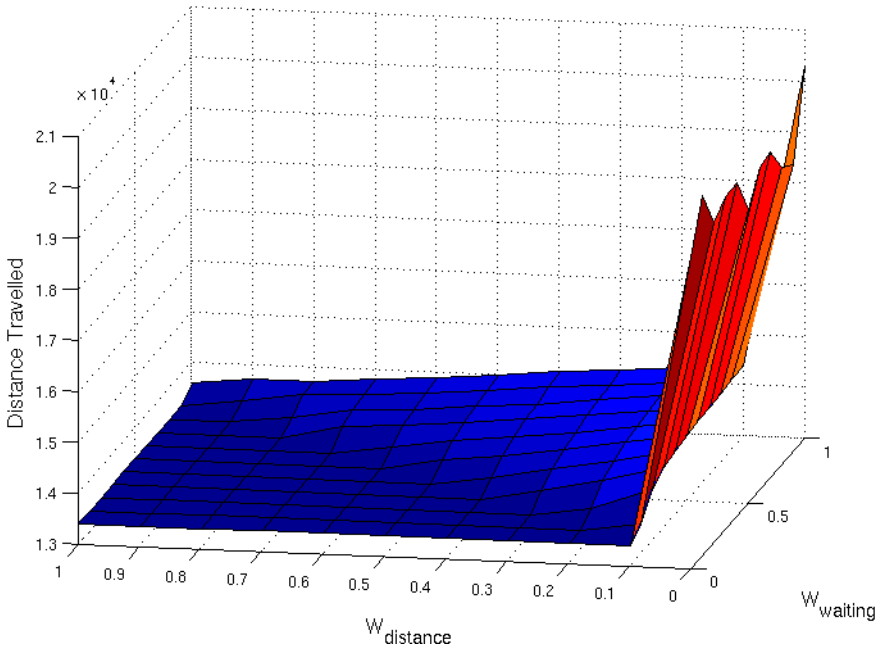

Figure 1.8: Performance landscape for $w_{waiting}, w_{distance}$ (distance travelled)

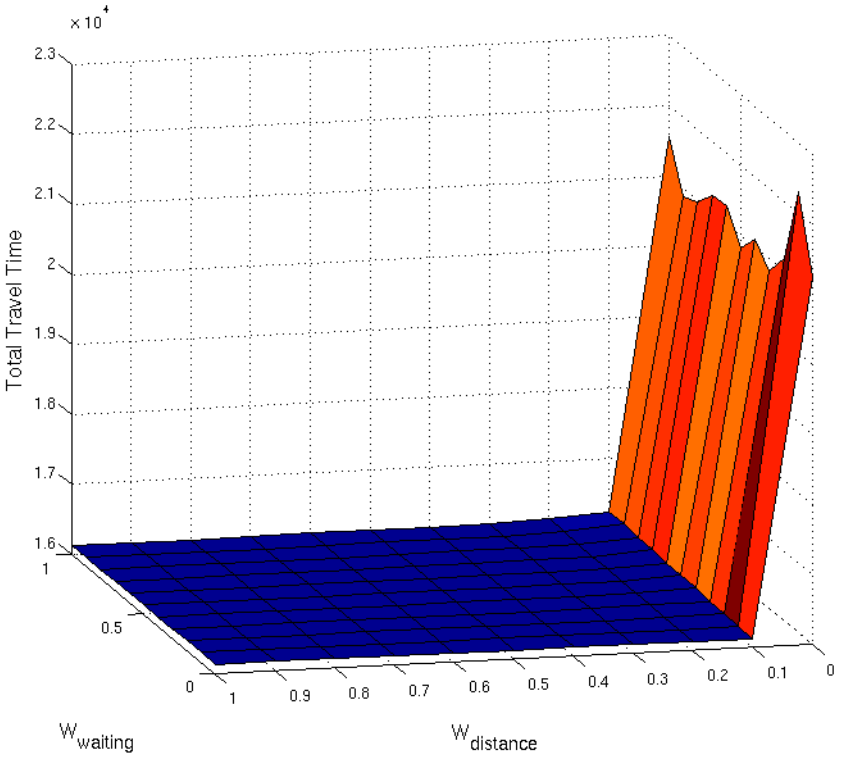

Figure 1.9: Performance landscape for $w_{waiting}, w_{distance}$ (total traveling time)

Again the results of the figures are intuitive: the more emphasis you place on congestion avoidance the less time you will spend waiting in queues. The downside is that as a result you will have to travel further and your trip might take longer. One may question, though, why the increase in total travel time is so extreme as it is for low values of $w_{distance}$ in figure 1.9. The reason is quite simple. The scenario only contains one bottleneck (E) and the only way to avoid it is by taking a very large detour (over H or over B and F). Therefore if the congestion at E is really bad (of equivalently, we increase $\frac{w_{waiting}}{w_{distance}}$) the cars have to travel a lot further to avoid the congestion (see figure 1.8). Consequently their total travel time is increased significantly. If we add a few extra edges or reduce the distances between D,H and G for example the results are not so extreme.

So, in sum, the plots show us that by varying the weights the TCS will optimize a different objective. Ultimately it will depend on the user of the TCS. Depending on priorities, the weights could be varied as follows :

- if the objective is to minimize travelling time, then distance and time standing still are given equal weight
- if the objective is to minimize fuel consumption, distance is given greater weight
- if the objective is to minimize frustration, which is most likely to occur when standing still, then congestion is to be avoided as much as possible, at the cost of increasing the distance

If we fix both $w_{waiting}, w_{distance}$ to 1 we can now investigate the impact of the other two parameters: ϵ (the evaporation rate) and δ (intention revision sensitivity). It is expected that decreasing the evaporation rate will disturb the congestion control, because reservations will be kept on for longer than they are needed. On the other hand, decreasing delta will make the cars more bold, they will change their intentions more quickly. This should make the congestion control more chaotic, since cars are liable to change route as soon as they find a marginally better route, and so reservations will change at a moment's notice. This, combined with an longer evaporation time (larger ϵ), could cause the congestion control to actually worsen the traffic management. Conversely, increasing the evaporation rate and making the cars more cautious, could have beneficial effect, up to a certain point, where cars don't ever change their intention about a route, even when they should.

The results are shown in figures 1.10, 1.11 and 1.12.

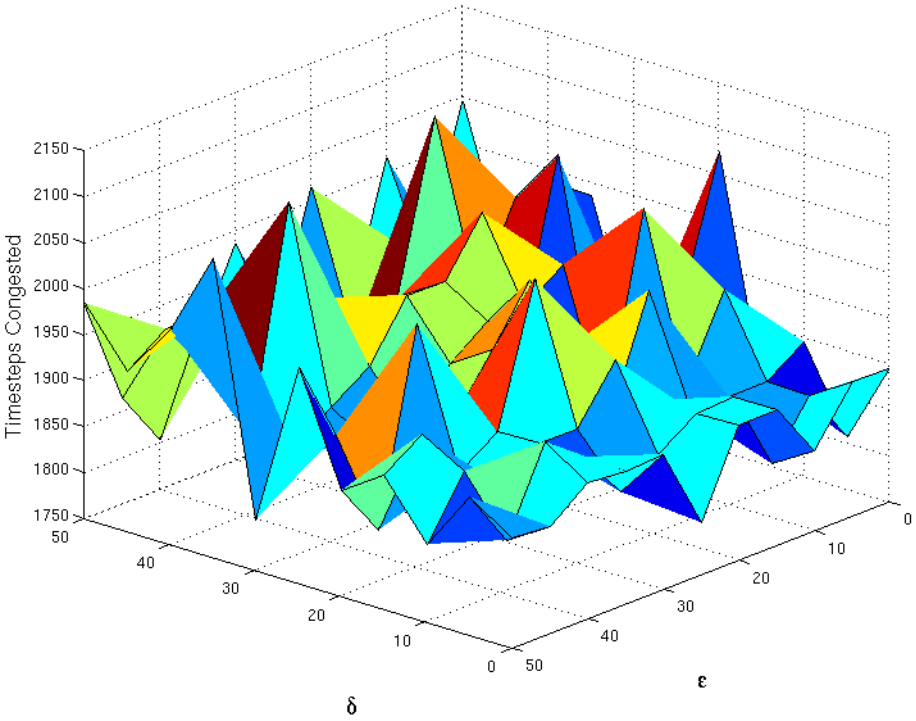

Figure 1.10: $\delta - \epsilon$ Performance Landscape (time congested)

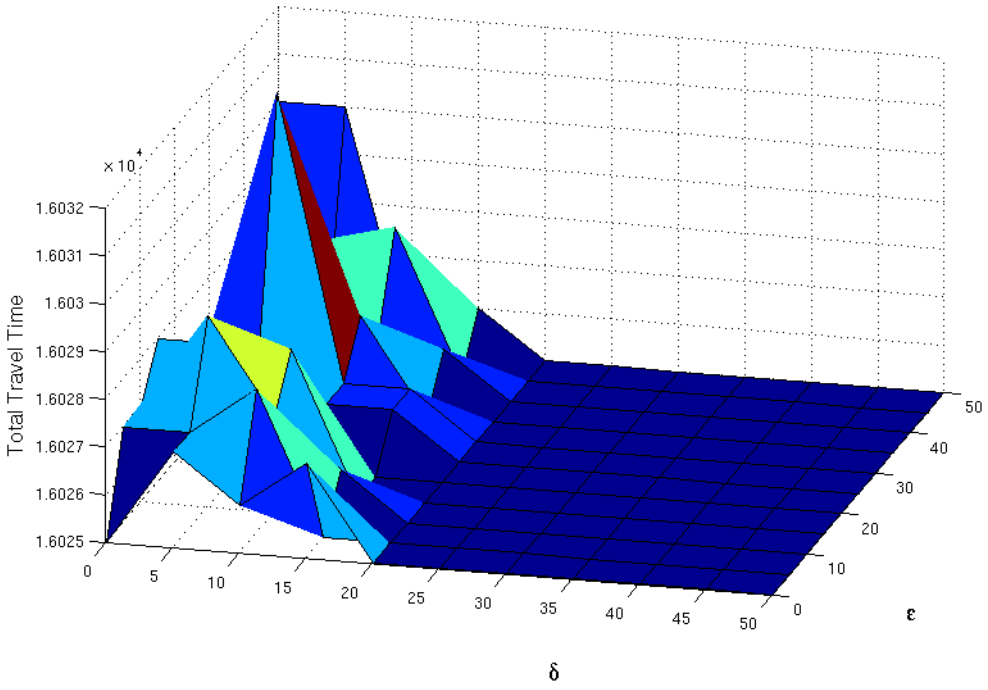

Figure 1.11: $\delta - \epsilon$ Performance Landscape (total travel time)

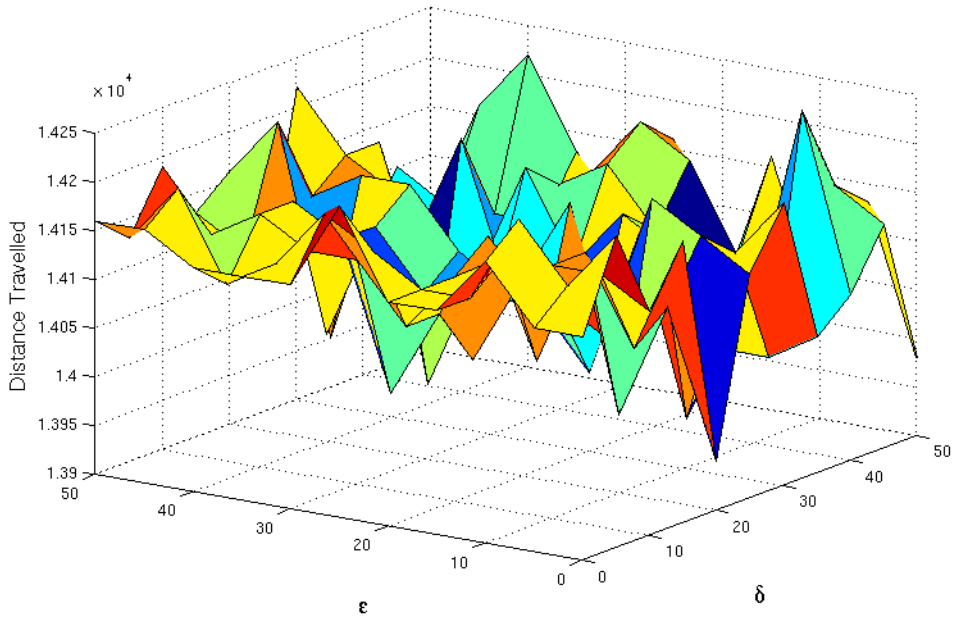

Figure 1.12: $\delta - \epsilon$ Performance Landscape (distance travelled)

Unfortunately it turns out that we don't really get what we would expect. The performance landscapes are very chaotic and it is hard to discern any trends. Only the total travel time landscape really agrees with the intuition that the faster we revise our intentions and the longer these intentions are reflected in the environment the worse the travel time. Further tests did not clarify this picture. It seems there is a complex interaction between scenario type, number and ordering of cars, δ and ϵ that defies any simple intuitive explanation.

Maybe the problem lies in the nature of the scenario: there are only a few 'choice points', and once those points are passed, not a lot of variation is possible. So the variation of boldness of agents and evaporation rate only counts in the planning before those choice points. When the evaporation rate is slow, the reservations, once made, will stay on beyond their usefulness, so this will distort the choice of the best path. Shortest paths will quickly become more expensive because more cars will initially choose them. If the agents are quite cautious, this influences their behaviour less, because once a path is chosen, this is less likely to change. This results in peaks and valleys for certain combinations of δ and ϵ, depending on whether the configuration allows them to make the right decision regarding congestion or distance before the critical choice points.

Combining congestion and distance into traveling time, the overal trend is clearer: beyond a certain value of cautiousness δ, the decision will not be that much influenced by the value of the evaporation rate ϵ, because the paths will not vary that much, so the evaporation rate doesn't influence the result. For bolder agents, decreasing the evaporation rate (increasing ϵ) will push up the traveling time, as the paths are more likely to change, and the 'old' reservations are more likely to interfere in the choice of the optimal path.

Logically the evaporation rate will be kept high (ϵ low), since there is no reason in this system to keep reservations alive longer than necessary.

1.4 Conclusion

In this project we have introduced the problem of traffic control and described the software architecture of a simulator that simulates traffic control. Due to the large scale and dynamic environment that characterizes traffic control (it is an example of a coordination and control problem) a Delegate-MAS based software architecture was chosen.

The TCS was implemented and its performance compared against a simple TCS that always routes the cars along the shortest path. As the previous section showed the use of pro-active congestion control proved beneficial and we can consider the implementation to work successfully. Unfortunately we were unable to properly interpret the relationship between the tuning parameters δ and ϵ. This is an issue that requires further investigation.

We now take a step back and consider critically what limitations are of our TCS and how the design and implementation may be improved. The major critique of our TCS is its simplicity. By adopting the simplifications listed in the requirements we sidestep many problems that arise in a real life traffic situation. However, considering the complexity of the problem a more realistic version would be out of scope for this project. Besides these simplifications, additional constraints of our implementation include:

- No backtracking ('U-turn') possible.
- Totally deterministic behavior. It would be interesting to study the effect of a stochastic decision process, where a car agent could choose a second-best or third-best road with a lower probability.

- Our current implementation does not rely on feasibility ants for mapping the road network. It is assumed that every car has navigational system that, given the network map, is able to return all possible paths between two places. Since all cars share the same network object (singleton) we *are* able to cope with dynamically changing networks, however our solution is not at all scalable to larger networks. There enumerating all possible paths explicitly is not feasible. However we stress that adding this functionality would be easy to do given our architecture. It would simply boil down to adding an extra method `Network.explore()` to the `main` method that would take care of sending out feasibility ants. The method `gatherPossibleRoutes(currentLoc, dest)` would then return the information gathered by the feasibility ants instead of a doing a full NP-complete graph search.
- The inability of being able to visualize the simulation in real time made it difficult to understand what was going on in more complex scenarios. This also made it difficult to see whether certain results were due to the congestion control mechanism or due to the particular scenario (order and placement of cars, particular network, ...).
- The system could use additional optimizations, running complex scenarios with many cars proved very computationally expensive.
- We did not consider communication among task agents (other than the indirect communication through ants), it would be interesting to investigate the impact of letting cars share knowledge in a P2P like manner.

Reflecting upon the Delegate-MAS software architecture we feel it to have been a good choice. It provides a nice balance between reactive and practical agents and has a natural implementation within traffic control. However, we do feel that one potential problematic aspect of the architecture is the definition of the feasibility ants. Defining how these ants should collect, or more importantly combine, feasibility information efficiently (avoiding flooding) in complex, dynamic graphs so that the information is truly useful does not seem an easy task. In addition a Delegate-MAS system may be hard to tune. Due to the many interacting agents it is hard to see what tuning parameter values make sense a priori. However in a sense this non determinism could be seen as a disadvantage of MAS in general. This does not remove the fact that it is a promising technology, and our simplified system gives a good enough result to warrant further investigation.

Bibliography

[1] Eclipse java development platform. http://www.eclipse.org/.

[2] Siemens road traffic technology. http://www.roadtraffic-technology.com/contractors/traffic_man/siemens_its/.

[3] Tno automotive - new transport systems. http://www.tno.nl/industrie_en_techniek/mobiliteit_en_(transport)/nieuwe_transportsystemen/.

[4] Bernd Bruegge and Allen H. Dutoit. *Requirements Elicitation*. Prentice-hall/pearson edition, 2004.

[5] Kurt Dresner and Peter Stone. Multiagent traffic managment : An improved intersection control mechanism. In *AAMAS'05*, Utrecht, Netherlands, July 2005. ACM.

[6] Tom Holvoet and Paul Valckenaers. Exploiting the environment for coordinating agent intentions. In *The Third International Workshop on Environments for Multiagent Systems (E4MAS2006)*, 2006.

[7] Craig Larman. *Applying UML and Patterns (Third Edition)*. Prentice hall ptr edition, 2004.

[8] Victor R. Lesser and Lee D. Erman. Distributed interpretation: a model and experiment. pages 120–139, 1988.

[9] N. R. Jennings M. Wooldridge. Software engineering with agents : Pitfalls and pratfalls. *IEEE Internet Computing*, 3:20–27, 1999.

[10] Bogdan Tatomir, Ronald Kroon, and Léon J. M. Rothkrantz. Dynamic routing in traffic networks using antnet. In *ANTS Workshop*, pages 424–425, 2004.

[11] Luis A. Garcia Vicente R. Tomas. A cooperative multiagent system for traffic managment and control. In *AAMAS'05*, Utrecht, Netherlands, July 2005. ACM.