Multi-agent transportation simulation

Kai Nagel

February 2, 2004

Contents

I Introduction 1

1 Introduction 1-1

2 A quick tour 2-1
- 2.1 Introduction . . . 2-1
- 2.2 Demand generation . . . 2-1
- 2.3 Traffic simulation . . . 2-2
- 2.4 Feedback . . . 2-3
- 2.5 Analysis . . . 2-3

II A do-it-yourself simulation package 2-5

3 Motivational start: Roundabout 3-1

4 Some basics of object-oriented programming 4-1
- 4.1 Introduction . . . 4-1
- 4.2 Compilation of programs under Unix . . . 4-1
- 4.3 Pointers . . . 4-2
- 4.4 Structs . . . 4-2
- 4.5 Classes and minimal memory management . . . 4-2
- 4.6 Encapsulation . . . 4-3
- 4.7 Constructors . . . 4-3
- 4.8 Arrays of classes . . . 4-3
- 4.9 The Standard Template Library (STL) . . . 4-4
- 4.10 Associative arrays/maps . . . 4-4
- 4.11 Methods; Inlining . . . 4-5
- 4.12 References (“&”) in subroutine calls . . . 4-5
- 4.13 “.” vs. “->” . . . 4-6
- 4.14 General code structure . . . 4-6
- 4.15 Review . . . 4-7

5 Some programming recommendations 5-1
- 5.1 General . . . 5-1
- 5.2 Programming language . . . 5-1
- 5.3 Compiler error messages for STL code . . . 5-2

5.4 Iterators . . . 5-2
5.5 Tokenizer . . . 5-3

6 Street network data and data structures 6-1
6.1 Introduction . . . 6-1
6.2 Network file formats . . . 6-2
6.3 Node class . . . 6-3
6.4 SimWorld class . . . 6-4
6.5 Nodes input . . . 6-4
6.6 Link class . . . 6-5
6.7 Links input . . . 6-5
6.8 Incoming/outgoing links . . . 6-6

7 Cellular automata micro-simulation 7-1
7.1 Introduction . . . 7-1
7.2 Vehicles . . . 7-1
7.3 Vehicles on links . . . 7-1
7.4 Random moves through intersections . . . 7-3
7.5 Fairer intersections . . . 7-4
7.6 Initializing vehicles for testing purposes . . . 7-4
7.7 Main program . . . 7-4

8 Visualizer 8-1
8.1 Introduction . . . 8-1
8.2 Vehicle output . . . 8-1
8.3 Visualization via gnuplot . . . 8-4
8.4 Testing the current status of the simulation . . . 8-4

9 Plans following in the micro-simulation 9-1
9.1 Plans . . . 9-1
9.2 Vehicle class . . . 9-2
9.3 Plans format . . . 9-2
9.4 ReadPlans . . . 9-4
9.5 Class Plan . . . 9-5
9.6 Park queue . . . 9-6
9.7 Wait queue . . . 9-7
9.8 Vehicle insertion . . . 9-7
9.9 Plans following and vehicle arrival . . . 9-8
9.10 Computational Speed . . . 9-9
9.11 Events output . . . 9-10

10 Modularization, inheritance, templates, and code re-use 10-1
10.1 Introduction . . . 10-1
10.2 Links, Simlinks, and Inheritance . . . 10-1
10.3 Templates . . . 10-2
10.4 What belongs into the base class? . . . 10-4

11 Route planner 11-1
11.1 Introduction . . . 11-1

11.2 Fastest Path . . . 11-1
11.3 Link travel times . . . 11-2
11.4 Library support for graph algorithms . . . 11-2
11.5 General structure . . . 11-2
11.6 Input file: Trips . . . 11-3
11.7 FindPath and Dijkstra . . . 11-4
11.8 Plans output . . . 11-6

12 Congestion-dependent router **12-1**
12.1 Link travel times and congestion . . . 12-1
12.2 Congestion dependency: Link travel times . . . 12-2

13 Feedback/System integration **13-1**
13.1 Introduction . . . 13-1
13.2 Subset of trips file . . . 13-1
13.3 Calling the router . . . 13-2
13.4 Merging of the routes . . . 13-3
13.5 Traffic simulation . . . 13-3
13.6 Iterations . . . 13-3

14 Activities planner: Adjust trip starting times **14-1**
14.1 Introduction . . . 14-1
14.2 Utilities . . . 14-1
14.3 Departure time selection . . . 14-3
14.4 Operationalization . . . 14-3
14.5 Input data: Activities file . . . 14-4
14.6 Origin-destination travel times . . . 14-4
14.7 Departure time choice . . . 14-5
14.8 Feedback . . . 14-6

15 Do-it-yourself transportation planning simulation: Summary **15-1**

16 File formats summary **16-1**
16.1 Nodes file . . . 16-1
16.2 Links file . . . 16-1
16.3 Snapshot file (visualizer output) . . . 16-2
16.4 Plans file . . . 16-3
16.5 Events file . . . 16-3
16.6 Trips file . . . 16-4
16.7 Activities file . . . 16-4

III Improvements 16-5

17 More realistic CA traffic simulation logic **17-1**
17.1 Introduction . . . 17-1
17.2 The stochastic traffic cellular automaton (STCA) . . . 17-1
17.3 Some validation of the STCA . . . 17-3
17.4 Lane changing . . . 17-4

17.5 Validation of lane changing rules 17-6
17.6 Traffic signals 17-6
17.7 Validation of traffic signal rules 17-7
17.8 Unprotected turns 17-7
17.9 Validation of rules for unprotected turns 17-8
17.10 Discussion 17-9

18 The queue model for traffic dynamics **18-1**
18.1 Introduction 18-1
18.2 General 18-1
18.3 Fair intersections 18-3
18.4 Limitations of the queue model 18-4

19 Routing **19-1**
19.1 Time aggregation 19-1
19.2 Generalized cost functions 19-1
19.3 Alternative routes 19-1
19.4 Logit for routes 19-2
19.5 Planning for given arrival time 19-2
19.6 Mental maps 19-3

20 Non-car modes of transportation **20-1**
20.1 Routing 20-1
20.2 Simulation 20-1

21 Demand **21-1**
21.1 Origin-destination matrices 21-1
21.2 Activities-based demand modeling 21-1

22 Feedback **22-1**
22.1 Introduction 22-1
22.2 Global trip times table 22-1
22.3 Agent data base 22-2
22.4 Day-to-day vs. within-day re-planning 22-3

23 Other Modules **23-1**

24 Better file formats **24-1**
24.1 Introduction 24-1
24.2 Use header line 24-1
24.3 XML 24-2
24.4 Some discussion 24-3

25 Parallel computing **25-1**
25.1 Introduction 25-1
25.2 Micro-simulation parallelization: Domain decomposition 25-1
25.3 Graph partitioning 25-2
25.4 Adaptive Load Balancing 25-5
25.5 Performance prediction for the Transims micro-simulation 25-8
25.6 Speed-up and efficiency 25-12

25.7 Other modules 25-16
25.8 Summary 25-16

26 Distributed computing and truly distributed intelligence **26-1**

IV Some background 26-3

27 Traffic flow theory **27-1**
27.1 Introduction 27-1
27.2 Traffic flow measurements 27-1
27.3 Fundamental diagrams 27-3
27.4 Car following 27-4
27.5 Kinematic waves and fluid-dynamics 27-12
27.6 Capacities, especially at bottlenecks 27-17
27.7 Cost-flow curves for static assignment 27-18

28 Static assignment **28-1**
28.1 Introduction 28-1
28.2 Equilibrium principle 28-1
28.3 Beckmann's mathematical programming formulation 28-3
28.4 Constrained optimization 28-3
28.5 Uniqueness 28-4
28.6 A solution method 28-5
28.7 Summary 28-7

29 Discrete choice theory **29-1**
29.1 Introduction 29-1
29.2 Binary choice 29-2
29.3 Multinomial choice 29-8
29.4 Discussion of modeling assumptions 29-9
29.5 Maximum likelihood estimation 29-10
29.6 Discussion 29-13
29.7 Summary 29-14

30 Axhausen lecture **30-1**

31 Learning and feedback **31-1**
31.1 Introduction 31-1
31.2 Additional aspects of day-to-day learning 31-1
31.3 Individualization of knowledge 31-2
31.4 Interpretation as dynamical system 31-4
31.5 Relation to game theory 31-8
31.6 Relation to machine learning 31-9
31.7 Smart agents and non-predictability 31-9
31.8 Conclusion 31-10

V Calibration and validation 31-12

32 Traffic flow characteristics 32-1

32.1 Introduction . . . 32-1
32.2 Validation, Calibration, etc. . . . 32-1
32.3 The Transims microsimulation approach . . . 32-3
32.4 Rules of the model . . . 32-4
32.5 Towards a standardized flow test suite for simulation models . . . 32-12
32.6 Yield sign behavior . . . 32-15
32.7 Comparison to Case Study Logic . . . 32-16
32.8 Short discussion . . . 32-16
32.9 Summary and conclusion . . . 32-17

33 Intersection test suite 33-1

34 Routing 34-1

35 A Dallas case – do I want this?? 35-1

36 A Portland/Oregon case 36-1

36.1 Introduction . . . 36-1
36.2 Problem statement . . . 36-2
36.3 Our approach . . . 36-3
36.4 Related work . . . 36-5
36.5 Experimental setup and simulation results . . . 36-6
36.6 Comparison to field data and to emme/2 study results . . . 36-8
36.7 Discusssion . . . 36-9
36.8 Summary . . . 36-12
36.9 Acknowledgments . . . 36-13

37 A Switzerland case 37-1

Part I

Introduction

Chapter 1

Introduction

Urban planning is not easy: People simultaneously want to have access to transportation and not be bothered by it. This is a contradiction which is not easily resolved, in particular not in densely populated areas. Urban and transportation planning are the disciplines which deal with this contradiction.

Any software package designed to help with these questions needs to address the fact that humans are "intelligent", that is, they are able to adapt and to learn. The maybe most prominent example in the realm of transportation planning is called induced traffic – the fact that better streets or better train connections leads to more traffic. In consequence, transportation planning is *not* an exercise of how to best deal with a given and fixed demand, but it has to balance the interests of people using the transportation system with the interests of people suffering from it.

A good approach to such complex problems are multi-agent simulations. Multi-agent means that all entities of the simulation, in particular the travelers, are resolved individually, and that they have internal rules according to which they make decisions and move inside the synthetic, simulated environment. Such an approach became possible with the advent of modern computers, which process rule-based logic as fast as numerical operations. A big advantage of this agent-based, microscopic approach is that it can be, at least in principle, arbitrarily improved if it turns out to be not realistic enough in certain aspects. This is in stark contrast to aggregated methods, which eventually reach a level where small-scale effects cannot be represented. As an example, 200 cars with 200 different destinations on a road can only be represented by having these 200 different destinations listed somewhere in the system; there is no useful way to average over them. Clearly, a natural place to store this information is inside the agents.

We do however believe that, once one has accepted the microscopic or agent-based paradigm, one can start with rather simple models. The primary purpose of this book is to show that full transportation simulation packages can be coded by somewhat experienced programmers in relatively short time. Such a package does not only contain the traffic micro-simulation, which moves vehicles and travelers through the system, but also modules for route planning, for activity generation, and, most importantly, for human learning. It is not claimed that the resulting transportation simulation package is calibrated and validated and thus useful for policy questions, but it is certainly complete enough to do computational research with respect to methodological and computational questions, and it could be a starting point for a more realistic package. In particular, it is possible to replace the modules one by one by more realistic ones and still keep the structure of the whole system intact. This makes it possible to pull together the efforts of many different research or commercial groups towards a large scale realistic multi-agent transportation simulation.

This book is based on a one-semester class with 3 hours per week, which are approximately evenly distributed between lectures and guided lab work. In addition, depending on their programming skills, students put in a significant homework effort (what many of them enthusiastically to). The class covers most of this book; homework comes in particular from Part II. The book is written in a way that Part II should be self-contained, that is, a reader mostly interested in basic code development should find all relevent information in that part of the book. The other chapters provide additional material, in particular with respect to improvements, and with respect to theoretical background. The perspective throughout the book is computational, that is, theoretical developments without relevance to a computational implementation are kept to a minimum.

Chapter 2

A quick tour

2.1 Introduction

Transportation simulation packages consist of several modules. The most important modules for the purposes of this book are: demand generation, route generation, and the traffic simulation (Fig. 2.1). In addition, a feedback module provides the coupling between these. The following sections will give short introductions into each of these modules.

2.2 Demand generation

2.2.1 Trip generation

The demand generation module generates the demand for the transportation simulation system. Two important methods are: (i) origin-destination matrices, and (ii) activity-based demand modeling.

Origin-destination (OD) matrices are the more traditional method. OD matrices contain the number of trips from n starting points to n destinations; it is therefore an $n \times n$ matrix. These matrices can refer to arbitrary time periods. Until a couple of years ago, one typically used 24-hour time periods; these days, people often concentrate on "morning peak" and "afternoon peak" periods since the main direction of travel is obviously different between these periods.

In many situations, it is desirable to have information about demand generation that goes beyond OD matrices. In such situations, the more far-reaching method of activities-based demand modeling is an alternative. Here, the simulation includes models of human behavior with respect to the planning of a day. This includes where and when to eat, sleep, work, shop, etc. For example, a person may start the day at home, be at work at 8am, work for eight hours, go shopping which takes an hour, then be at home for the rest of the day. Assuming that all the transportation pieces take half an hour, this would fix the transportation schedule to: leave home at 7:30am, be at work at 8am, leave work at 4pm, arrive at shopping at 4:30pm, leave shopping at 5:30pm, arrive home at 6pm.

Once the simulation "knows" where and when people do their activities, transportation is generated via connecting activities that take place at different locations. Note that it is not necessary (and probably not possible) to forecast such activities for specific persons; however, there is hope that we will be able to get useful ensemble averages similarly to Statistical Physics.

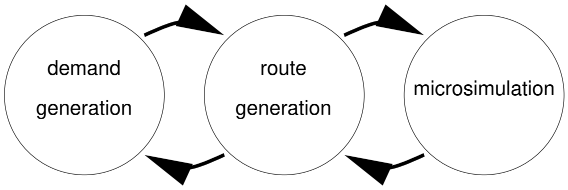


Figure 2.1: Modules

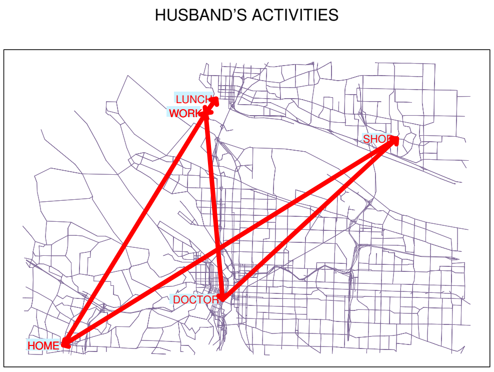


Figure 2.2: Illustration of a daily activity plan.

2.2.2 Route generation

Once trips (e.g. starting times, starting locations, and destination locations) are known, the exact transportation for these needs to be generated. This includes mode choice (walking, bicycle, train, car, etc.) and the precise routing. The output of this module are complete plans for each individual in the simulation.

2.3 Traffic simulation

Now these plans need to be executed. These simulations come at many different levels of resolution and fidelity, reaching from the traditional steady-state flow-based cost function to very detailed micro-simulations.

If one is interested in time-dependent results, as for example the queue built-up during the onset of rush periods, the simulation needs to be sufficiently realistic to contain such dynamics. Traditional flow-based cost functions are *not* able to realistically deal with such dynamical effects, at least not in a straightforward way. Thus, the right simulation has to be chosen according to what aspects of the dynamics one wants to have represented for a given question.

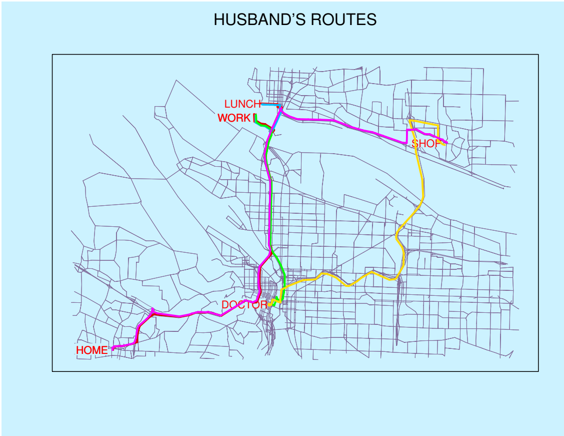


Figure 2.3: Illustration of a daily plan including routes.

2.4 Feedback

The traffic simulation needs input from the demand generation, since it executes the plans from the demand generation. However, the demand generation depends on the traffic simulation because for example congestion only shows up in the traffic simulation, and demand adjusts to such shortages. In order to deal with this situation, one iterates between demand generation and traffic simulation. For example, demand generation is run assuming no congestion, the resulting traffic simulation is run, then the demand generation is run again now including the congestion from the last traffic simulation run, etc., until a steady state is reached. That is, the system is systematically relaxed towards a consistent state.

Fig. 2.4 shows an example of replanning. The traveler first changes his/her route, presumably in adaptation to congestion. Eventually, he/she decides that the destination is too far away and switches to a nearer location. Fig. 2.5 shows a systemwide consequence of replanning. The scenario is one where 50 000 travelers starting at random locations all over Switzerland travel to Lugano, which is south of the Alps. The scenario is for testing purposes, but it has some resemblance with vacation traffic in Switzerland. In the initial run (left), all travelers have planned their routes assuming a completely empty network; in consequence, they all use the freeways as much as possible. After many iterations (right), travelers have learnt that because of the congestion other paths may be advantagous; as a result, traffic is much more spread out.

It should be noted at this point that there is no a priori reason why a real system should be relaxed. For example, during unique events such as trade shows or soccer games, the transportation system is probably not relaxed. The research here just follows the usual path in such situations: First understand the steady state solution, and then move on to the transients. Note that the steady state here refers to the comparison from one iteration to the next, *not* to a steady state across time-of-day.

2.5 Analysis

Once a representative run or collection of runs of the traffic simulation has been obtained, it can be analyzed. For example, one can see where congestion will show up, and which people get stuck in it. Analysis is the other aspect of the system that influences the decision about the level of realism in the modules. For example, if one is interested in

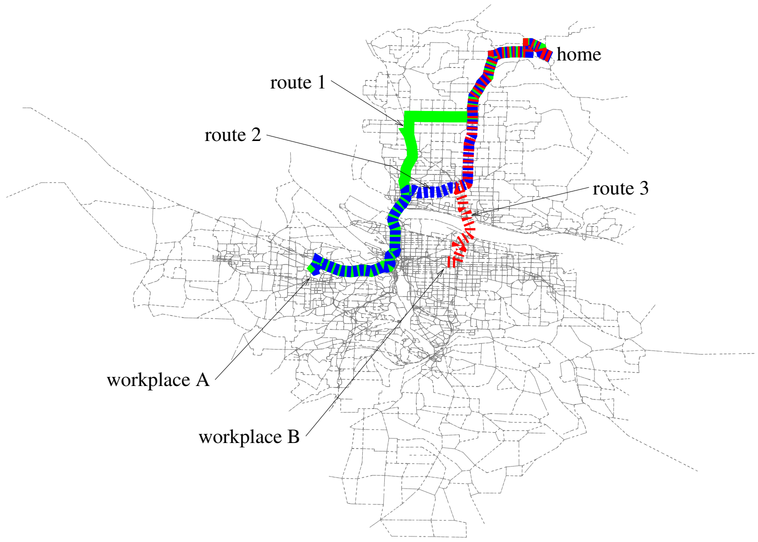


Figure 2.4: Result of day-to-day learning in a test example. LEFT: Situation at 9:00am in the initial run. RIGHT: Situation at 9:00am in the 49th iteration. Each pixel on the road is a car (by overlapping in the graphics they form the traffic streams); the circle denotes where they are going. Clearly, the system has found a better solution after 49 iterations.

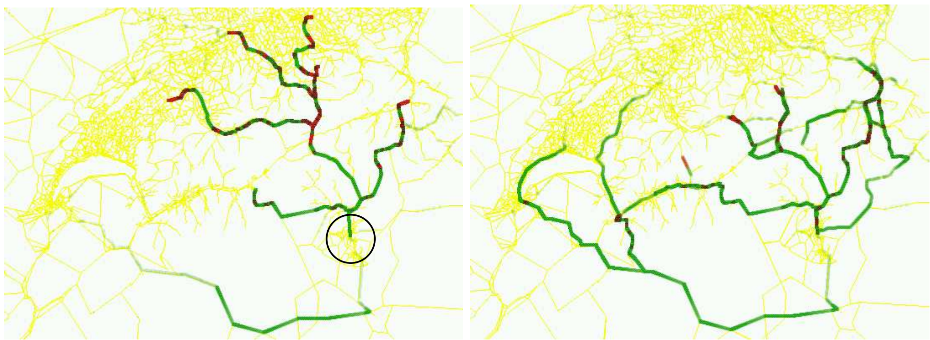


Figure 2.5: Feedback

emissions, one needs a micro-simulation of the driving behavior with enough information on, e.g., acceleration in order to derive the necessary quantities. Or if one is interested in the possible rescheduling of activities as a consequence of transportation infrastructure changes, one needs to model the effect of "trip chaining", i.e. the fact that people can for example go shopping on the way back from work, but they could also put in a stop at home before they go shopping.

Part II

A do-it-yourself simulation package

Chapter 3

Motivational start: Roundabout

In this chapter, we will consider the question if for an intersection it is better to have traffic lights or a roundabout. Our model is the simplest version that makes some sense.

The purpose of this chapter is to familiarize the reader with the general thinking that is used throughout this book: Models are started from simple first principles. In the following model, as in all models introduced in this book, the reader will easily detect imperfections. It is left to the curious reader (and programmer) to implement and test improvements.

We consider an intersection with four incoming/outgoing streets (Fig. 3). Streets are numbered 0, 1, 2, 3 as shown in the picture. We only model the incoming streets; as soon as vehicles leave the roundabout or the intersection, they have left our simulation world.

At each incoming streets, vehicles enter the simulation randomly but with a fixed rate. Each incoming vehicle selects any of the outgoing links as destination, excluding its own link.

Vehicles are moved forward along the link using the so-called cellular automata (CA) technique. This technique partitions space into cells which are updated via simple rules. In our situation, the street will consist of cells which are either empty, or occupied by exactly one vehicle. The system uses a parallel update (Fig. 3): All vehicles that have an empty cell in front of them at time t can move one cell; the result is the configuration for time $t+1$. Vehicles at the end of the link can only continue when the traffic light is green, or when there is space on the roundabout.

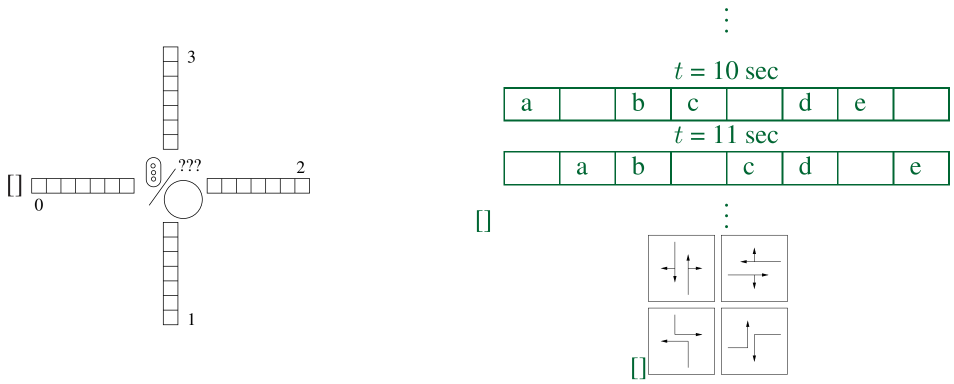


Figure 3.1: (a) Schematic drawing. (b) Cellular automata driving logic. (c) The four traffic light phases.

The traffic light

The traffic light has four phases as indicated in Fig. 3. There are no "yellow" times between the phases (although they can be introduced easily). Vehicles can enter the intersection if the traffic light allows them to go into the direction desired by the vehicle. Otherwise, the vehicle will stop, blocking all other vehicles behind. Vehicles that are allowed to enter the intersection are removed from the simulation, that is, there is no interaction of vehicles inside or beyond the intersection.

The roundabout **[[need fig]]**

The roundabout is modeled as a circular street, that is, it is a CA array of its own. Vehicles that leave the last array cell enter the first array cell. There are four entry cells into that circular array, corresponding to the four streets. A vehicle can enter when the entry cell and its upstream neighbor are empty. Vehicles leave one cell before the corresponding entry cell.

Implementation

Many possibilities exist to implement this, and experienced programmers will find there own system. The following paragraphs will provide some guidance, but they will not replace a programming class.

The programming style selected in this chapter is the most basic one we could think of. Later chapters will progressively introduce somewhat more advanced concepts.

CA links

The four CA links can be implemented as

```
const double RATE=0.2 ;
const int LL=10 ;
const int NN=4 ;
int cells[LL][NN] ;
int tmpcells[LL][NN] ;
const int EMPTY=-1 ;
...
// go through time:
for ( int tt=0; tt<TT; tt++ ) {
    // go through all streets:
    for ( int nn=0; nn<NN; nn++ ) {
        // enter a vehicle if this is possible:
        if ( cells[0][nn] == EMPTY && drand48() < RATE ) {
            // select a number between 0 and NN-2:
            int destination = int( (double)(NN-1) * drand48() ) ;
            // if self is selected, use NN-1:
            if (destination==nn) { destination = NN-1 ; }
            tmpcells[0][nn] = destination ;
        }
        // go through all cells except last:
        for ( int ii=0; ii<LL-1; ii++ ) {
            if ( cells[ii][nn] != EMPTY && cells[ii+1][nn]==EMPTY ) {
                tmpcells[ii+1][nn] = cells[ii][nn] ;
            } else {
                tmpcells[ii][nn] = cells[ii][nn] ;
            }
        }
        // special treatment for last cell:
        if ( intersection_can_be_entered ) {
            move_vehicle_to_intersection ;
        }
    }
    // copy tmp array back to main array and clear tmp array:
    for ( int nn=0; nn<NN; nn++ ) {
        for ( int ii=0; ii<LL; ii++ ) {
            cells[ii][nn] = tmpcells[ii][nn] ;
            tmpcells[ii][nn] = EMPTY ;
        }
    }
}
```

Traffic signal Again, there are many ways to implement this. Let us, for simplicity, assume that each of the `NPHASES` phases takes `PP` seconds; the phase is then given by

```
for ( int tt=0; ... ) {
    int phase = (tt/PP) % NPHASES ;
}
```

where `%` is the C++ modulo operation. Let us then define a function

```
bool allowed ( int from, int to, int phase )
```

which returns `true` when movement from link `from` to link `to` is allowed in phase `phase`, and `false` otherwise. Intersection movement can then be modeled as

```
        // special treatment for last cell:
        if ( cells[LL-1][nn]!=EMPTY ) {
            int destination = cells[LL-1][nn] ;
            // if movement NOT allowed, keep vehicle:
            if ( !allowed( nn, destination, phase ) ) {
                tmpcells[LL-1][nn] = cells[LL-1][nn] ;
            }
        }
```

Roundabout Implementation of the roundabout is left to the creativity of the reader. Note that there are some subtle timing issues involved: A reasonably clean implementation should not allow a vehicle to move two cells in a given time step; this would mean that a vehicle that just entered the roundabout is not allowed to make another move inside the roundabout. This can be achieved by first computing the `tmpcells` for *all* links, and only then copying them back to `cells`. In that way, a vehicle entering a roundabout would be copied into the `tmpcells` of the roundabout, where it would not be moved any further during the time step. Obviously, one has to be careful that no other vehicle overwrites this vehicle in `tmpcells`.

Output Experienced programmers will have their preferred visualization toolkit. Here we just want to point out that, to a certain extent, it is possible to derive graphics from simple terminal operations. For example, links can be plotted by

```
#include <iostream>
...
for ( int ii=0; ii<LL; ii++ ) {
    if ( cells[ii][nn] != EMPTY ) {
        // if there is a vehicle, output its destination:
        cout << cells[ii][nn] ;
    } else {
        // else output an empty space:
        cout << " " ;
    }
}
// Don't forget the newline once the link is plotted:
cout << endl ;
```

Most platforms have a so-called vt100 terminal; under unix this can often be obtained by typing `setenv TERM vt100` in an xterm. For example, the command

```
    cout << "\033[H\033[2J" ;
```

erases the screen, allowing the program to overwrite what was there before. This makes it possible to display the complete intersection dynamics as a movie inside a text terminal.

Variations As said before, this is a very simplistic model, and many modifications of this are possible. Some examples:

- The link lengths, the entry rates, the signal phases, or the size of the roundabout could be changed. Signal phases could be made adaptive.
- The entry conditions into the roundabout can be changed.
- There could be separate lanes for left turns. How long should they be?
- There could be inhomogeneous demand.
- Etc.

Chapter 4

Some basics of object-oriented programming

4.1 Introduction

We attempt to use relatively "lightweight" object-oriented programming. However, unfortunately this depends on the perspective and experience. I hope that even someone without experience will be able to get the most important things done. However, some solid programming experience is most probably helpful. If you have never seen pointers or structs/classes, it is going to be hard.

Before you get desperate, maybe have a look at Sec. 4.15 to see how (relatively) easy it will be at the end.

Implementation

4.2 Compilation of programs under Unix

If you are an unexperienced programmer, I recommend to write everything into one file, say `work.cpp`. This is then compiled with

```
g++ work.cpp
```

and executed with

```
./a.out
```

You need at least `g++` version 2.96; the version number can be found out by the command `g++ -v`.

You should put the following lines at the beginning of `work.cpp`:

```
#include <assert.h> // assert macro; see ``man assert''
#include <iostream> // cin/cout
#include <math.h>
#include <stdlib.h>
```

If you are using a Microsoft compiler, the following may help:

```
#if _MSC_VER > 1020              // if VC++ version is > 4.2
        using namespace std;     // std c++ libs implemented in std
#endif
```

The following should print "hello world" once:

```
// put above headers here
int main() {
    cout << "hello world" << endl ;

    return 0 ;
}
```

4.3 Pointers

At first, one typically does things such as

```
int id = 1 ;
double xCoord = 2.34 ;
cout << id << endl ;
cout << xCoord << endl ;
```

Pointers allow to put the real stuff somewhere else and to reference it by an address:

```
int* id ; *id = 1 ;
double* xCoord ; *xCoord = 2.34 ;
cout << *id << endl ;
cout << *xCoord << endl;
```

What this means is that `id` itself contains just a memory address, and the real content is where this memory address points to. `*(...)` can thus be read as "contents of `(...)`".

This does not have any advantage at this level; but it has enormous advantages as soon as the content that the memory address points to is more than a simple number.

4.4 Structs

Plain C allows things like

```
struct Node {
    int id ;
    double xCoord ;
    double yCoord ;
};
```

This means that our node has properties, such as an ID number and coordinates. These are used as follows:

```
struct Node node ;
...
node.id = 213 ; //assingment of ID number 213
xx = node.xCoord ; // retrieval of xCoord
```

Typically, this is however used in pointer syntax; the example then is

```
// this does not work yet, see text
struct Node* node ;
...
node->id = 213 ;
xx = node->xCoord ;
```

Note that the arrow `->` comes from converting `Node node` into `Node* node`. That is, arrows mean that the thing *to the left of them* is a pointer.

There is not yet a big advantage of using it this way. If one looks at the memory management, then `struct Node* node` only reserves space for the memory address itself; we would however also need memory space for `id, xCoord, yCoord`, which we don't have at this level. This will be solved in the next paragraph.

4.5 Classes and minimal memory management

In C++, we can replace `struct` by class:

```
class Node {
    int id ;
    double xCoord ;
    double yCoord ;
};
...
Node* node ; // reserve space for memory address
...
node = new Node() ; // reserve memory space for contents
...
node->id = 213 ;
xx = node->xCoord ;
```

the use of `new` also solves the memory problem.[1]

4.6 Encapsulation

In C++, one typically encapsulates variables. This does not have a major advantage at the level of this text, but we do it to conform with the standard. It goes as follows:

```
class Node {
private:
    int id_ ; // Convention: I add underscores to private variables.
    double xCoord_ ;
    double yCoord_ ;
public:
    void set_id( int tmp ) { id_ = tmp ; }
    int id() { return id_ ; }
    void set_x( double tmp ) { xCoord_ = tmp ; }
    double x() { return xCoord_ ; }
    ...
} ;
...
Node* node ;
...
node = new Node() ;
...
node->set_id( 213 ) ;
xx = node->x() ;
```

`private:` means that everything in that block can only be accessed by methods which are defined inside the class definition, i.e. inside the `class Node` block.

4.7 Constructors

"`new ...`" is also called "calling a constructor". In the above example, we have not defined what the constructor does; for this case, C++ provides a so-called default constructor. One can re-define the constructor, and one can even call it with arguments. Although that feature can lead to more robust code, we will not use it here.[2]

4.8 Arrays of classes

Typically, we have more than one node. The straightforward way to do this would be

```
...
Node* nodes[20] ; // allocate 20 memory addresses
...
nodes[0] = new Node ( ) ; // allocate space for ONE (!) node
```

[1] In C, this would be done via `malloc`.

[2] For experts: The main reason why we do not use it is because constructors are not inherited. For templatized classes, as will be useful for the network construction (Sec. 10), this means that each change of the constructor arguments in the template methods necessitates corresponding changes in all derived classes. We found that rather inconvenient.

```
...
nodes[0]->set_id( 213 ) ;
xx = nodes[0]->x() ;
```

4.9 The Standard Template Library (STL)

The above array usage is awkward because we need to know in advance how many nodes we will have. It is better to use vectors, as follows:

```
#include <vector>
...
vector<Node*> nodes ;
...
// memory management missing
...
nodes[0]->set_id( 213 ) ;
xx = nodes[0]->x() ;
```

So the usage of this looks the same as before, but the memory management is still missing. An easy way to enter elements without having to worry about memory is to use one of the insertion operators:

```
...
Node* node = new Node( ... );
nodes.push_back( node ) ; // add array element at end
...
```

It helps to use typedefs:

```
...
typedef vector<Node*> Nodes ;
Nodes nodes ;
...
```

(instead of `vector<Node*> nodes;`).

Note that now

```
Nodes nodes ;
```

essentially looks like and is used like

```
Node* nodes[20] ;
```

except that the memory management is different.

`vector<Node*>` is template syntax; it means that we have a vector of type `Node*`. Instead of vector, you could think "array".

Besides `vector`, there are other pre-defined template classes, such as `list` and `deque`. They all have certain insertion and removal operations which do the memory management for us. In C++, this is known as the Standard Template Library (STL). It is included in all new enough C++ compilers.

We will always hide templates via typedefs so in general they will not really show up. They do however (unfortunately) make a big difference in compiler error messages (see 5.3).

4.10 Associative arrays/maps

In C-arrays, one needs that indices start at zero and are consecutive. In transportation and many other areas, items such as nodes and streets have names or numbers. In our context, the nodes/links have numbers, and they are unique, but not consecutive. What we want is a data structure that deals with this in a straightforward way, i.e. where we can retrieve a node with ID "231" by `node[231]`. Associative arrays do this. They are used as follows:

```
#include <map>
...
typedef map<int,Node*> Nodes ;
...
Nodes nodes ;
...
// allocate space for new node and fill with information:
Node* node = new Node(id,xCoord,yCoord);

// register this node with the global nodes array:
nodes[id] = node;
...
```

Use of this now is:

```
cout << "ID:" << nodes[213]->id() << endl ;
cout << "X :" << nodes[213]->x() << endl ;
```

4.11 Methods; Inlining

We had already constructs like

```
class Node {
   ...
   double x() { return xCoord_ ;}
};
```

One can put arbitrary functions here, e.g.

```
class Node {
    ...
    intersectionLogic() {
        // lots of stuff
    }
};
```

This is called a method of the class. This version is the "inlined" version of the method.

Often, this gets so long that one wants to have this outside the class definition. In this case one would write:

```
class Node {
    ...
    intersectionLogic() ;
};
```

and somewhere else

```
Node::intersectionLogic() {
    // lots of stuff
}
```

Conventionally, one would put the first part into a `*.h` file, and the second part into a `*.cpp` file. It is however also possible to leave everything in `work.cpp`.

Inlined functions/methods are faster during the execution but need more memory and more compilation time.

4.12 References ("&") in subroutine calls

C and C++ by default call subroutine arguments "by value", which means that they copy the complete object. For example,

```
void doSomething( Nodes nodes ) {
...
}
...
   doSomething( theNodes ) ;
...
```

would copy the whole `Nodes` data structure and then operate on that copy. That has two often undesired or unexpected side-effects:

- The `Nodes` object can be rather large: For large road networks, it contains all pointers to all nodes.
- Changes in `Nodes` are not moved up to the main program.

This behavior can be avoided when *references* are used, as follows:

```
void doSomething( Nodes& nodes ) {
...
}
...
   doSomething( theNodes ) ;
...
```

Note the "`&`" in the argument list. The result of this is that `doSomething` will directly use the already existing nodes data structure.

In general, we will always use references in subroutine calls. Only when we pass `int` or `double` will we, wenn we do not want to pass back a result, omit the "`&`".

References can also be used in other contexts, in particular to avoid pointers to objects (see below). We will not use them for that since we find the pointer version easier to understand for non-experts.

4.13 "." vs. "->"

In the above, methods inside classes are addressed via the `->` operator. Sometimes, one has to use the `.` operator instead. Unfortunately, we are unable to write efficient code which uses consistently one or the other, so you need to understand the difference. That difference is that `x->y()` means that `x` is a pointer, while `x.y()` means that `x` is the object itself or a reference to it. As a rule of thumb, we will use "`->`" when we use objects, and "`.`" when we use containers. For example:

```
typedef map<Id,Node*> Nodes ; // Nodes contains *pointers* to Node!
Nodes nodes ; // nodes is *not* a pointer
...
for (Nodes::iterator nn=nodes.begin() ; // since ``nodes'' is not a
                     nn!=nodes.end()    // pointer, ``.'' is used.
                     ++nn ) {
Node* node = nn->second ; // ``node'' is now a pointer
...
cout << node->id() << endl ; // ``->'' is used
...
```

4.14 General code structure

Even if you write everything into one file, which simplifies life for non-experts, there is some structure that should be obeyed and that helps later to pull the code apart into several files. It is as follows:

```
// Global declarations/definitions.
// This would become something like ``globals.h''.
typedef double Time ;
Time time = -1 ;
...

// global utilities
// This would become something like ``utils.h''.
#include <stdlib.h>
extern "C" double drand48() ;
double myRand() {
    return drand48() ;
}
```

```
// Class declarations including definitions for ``short'' methods.
// Each class would go into a separate *.h file.
class Link ; // forward declaration
class Node {
private:
    Id id_ ;
public:
    void set_id( Id val ) { id_ = val ; } // ``short'' method
    ...
    Link* findOutgoingLink( Id linkId ) ; // ``long'' method
};
...

// Definitions of ``long'' class methods.
// Methods for each class would go into a separate *.cpp file.
Link* Node::findOutgoingLink( Id linkId ) {
    ...
}
...

// global functions (should be avoided; can normally go into ``class
// SimWorld'' or similar)

// main:
void main() {
    ...
}
```

4.15 Review

The most important information that you hopefully take from the above is that when you copy something like

```
#include <map>
...
typedef map<int,Node*> Nodes ;
...
Nodes nodes ;
...
Node* node = new Node( ... ); // allocate space for new node
...
```

from this text, then afterwards the use of this is relatively straightforward:

```
cout << "ID:" << nodes[213]->id() << endl ;
cout << "X :" << nodes[213]->x() << endl ;
```

Chapter 5

Some programming recommendations

Implementation

5.1 General

We recommend to use variable names which are easy to remember. We also recommend to write "robust" code, because this piece of code will be used over and over again, and it will be improved bit by bit. Robust means the following for me:

- Things which can go wrong need to be tested during execution and should lead to a program abort if the test fails. In my experience, warnings are not useful here since in the end there will be so many warnings that one will ignore them all. For example, one should test for memory boundaries. `assert()` is a useful C/C++ command, see `man assert`.
- As a *minimum* rule for the use of subroutines: Functionality which is used more than once inside a program has to go into a subroutine.

Personally, I think that for simulation problems the strict observation of these two rules are by far the most important aspects of structured programming. This is independent from the particular programming language; it is also independent from the object-orientedness of the programming language although it may help.

5.2 Programming language

Many programming languages are suitable to write traffic simulations. Here are some comments about the most common ones:

- C – "small" language; fast; objects are available via `struct` but no further object support; in general very little support for things that one needs for agent-based simulation
- C++ – "big" language that few people know completely (i.e. significant risk that one writes code that nobody can read); object-oriented language with decent support for agent-based simulation; good support for high performance computing in particular for object-oriented numerics; no standardized support for graphical user interfaces.
- java – similar to C++; includes support for graphical user interfaces. Well-written code in java is not necessarily slower than code in C++, but there is in general less

support for high performance computing (parallel compilers; debugging of parallel code; object-oriented numerics; ...).

- fortran – comes from the tradition of numerical analysis; newer versions of fortran have some support for agent-based simulation but no comparison to C++ or java

Recommendation: C++ or java, depending on own experience.

In the following, we will often give examples in C++ style. The goal is not to push C++ to its limits (as said above, in our experience very few people can read and maintain the resulting code) but to end up with design patterns that hopefully help average programmers. We will use the Standard Template Library (STL) where we feel that this is helpful.

5.3 Compiler error messages for STL code

Compiler error messages for STL code are awkward. Here is an example:

```
In file included from sim.cpp:5:
global.h: In function `void Simulate (int, map<id, Node *, less<Id>,
allocator<Node *> >, map<Id, Link *, less<Id>, allocator<Link *> >,
map<Id, Veh *, less<Id>, allocator<Veh *> >)':
global.h:358: conversion from `Link *' to non-scalar type `Link'
requested
```

It is often helpful to first read the messages item by item and sometimes to re-arrange the messages:

- First comes where the corresponding file was included:

  ```
  In file included from sim.cpp:5:
  ```

- This is followed by the function where the error happens:

  ```
  global.h: In function `void simulate (int, map<Id, Node *, less<Id>,
  allocator<Node *> >, map<Id, Link *, less<Id>, allocator<Link *> >,
  map<id, Veh *, less<Id>, allocator<Veh *> >)':
  ```

  As long as there is only one function `void simulate(...)`, one can ignore the rest of this part of the error message. If one does not know about function overloading, this should be generically the case.

- Finally comes the real error message. Rearranging yields:

  ```
  global.h:358: conversion from
    `Link *'
  to non-scalar type
    `Link'
  requested
  ```

  That is, somehow the item on the right is a pointer to link, while the item on the left is a link.

  In this case, the offending line was

  ```
  Link link = l->second;
  ```

  The correct line would be

  ```
  Link* link = l->second;
  ```

5.4 Iterators

Simulations often need to iterate over all objects in a certain class, for example over all agents or all streets.

In C++, iterators are explicitely provided for many data structures of the STL. Code typically looks like the following:

```
for (Links::iterator ll = links.begin(); ll != links.end(); ll++) {
    Link* link = ll->second ;
}
```

The `->second` is necessary if `Links` is, as discussed in Sec. 4.10, a `map<int,Link>`. Then `ll` returns the "pair" `(int,Link*)`, while `->second` just returns the second item. – This will be filled with more meaning in later examples.

5.5 Tokenizer

In order to read line-oriented input files, it is useful to first read the complete line (`getline`), and then to parse it. This can look as follows:

```
assert( inFile.is_open() ) ;
typedef vector<string> Tokens; Tokens tokens ;
while ( !inFile.eof() ) {
    string aString ; getline( inFile, aString ) ;
    if ( !aString.empty() ) { // ( skip empty lines )
        tokenize( aString, tokens ) ;
        for ( Tokens::iterator tt=tokens.begin() ; ii!=tokens.end() ; ii++ ) {
            cout << *tt << "\n" ;
        }
    }
}
```

As of 2003, there is unfortunately no standard tokenizer for C++. A simple tokenizer, which separates on white spaces (such as blanks and tabs), is the following (from the linux `C++-programming-howto`):

```
#include <iostream>
#include <fstream>
#include <sstream>
#include <string>
#include <vector>
...
inline void tokenize ( const string& str, vector<string>& tokens ) {
    tokens.erase( tokens.begin(), tokens.end() ) ;
    tokens.push_back( "TRASH" ) ; // do not use tokens[0] ;
    string buf ;
    stringstream ss(str) ;
    while( ss >> buf ) {
        tokens.push_back(buf) ;
    }
}
```

This is slightly modified when compared to the original version in so far as it puts "TRASH" into the zeroth element so that the counting of tokens starts with one. This has the advantage that a token from the nth column will be in `token[n]`.

Chapter 6

Street network data and data structures

6.1 Introduction

Transportation simulations need to deal with real world scenarios to be useful. In order to achieve this, it makes sense to write them so that they can read arbitrary real world configurations, even when the initial intention of the project is to use artificial data. For the example case of this text, the minimum content of the data base is some information about the road network, and some information about where people live and where people work.

In this section, the information about the road network is considered. The basis for this is a simple coding that is usually used for graphs, with one file/list for nodes (vertices) and one file/list for links (edges, arcs). The traffic network then is built by identifying links with roads, and intersections with nodes. Our intersections will be extremely simplistic.

The node file typically contains:

- a unique ID number for each node, and
- geographical coordinates.

Additional information can be added for each node, but is not needed for this example.

The link file for this example needs the following information:

- a unique ID number for each link,
- the ID number of the node where the link starts,
- the ID number of the node where the link ends,
- length of the link (length is necessary because a curvy road between two nodes will be longer than the Euclidean distance),

Implementation

6.2 Network file formats

The first implementation question to resolve is how to store the data. We will assume that the data is in a file, and that is uses the same format that the transportation simulation software package Transims (?) uses. Transims file formats are used several times in this text. The advantage is some degree of portability; the disadvantage is that the formats often contain many more entries than we truly need. Also, a more modern format might use some kind of XML syntax; there is however no corresponding standard for transportation simulations. We think that the advantage of using Transims files outweighs the disadvantages. XML formats will be discussed in Sec. 24.3.

Each Transims network file has a header line, and then zero or more lines of entries. The header line needs to be there; it contains the keys of the entries. Fields are separated by tabs.

The nodes file has the following entries:

Column	Header	type	explanation
1	**ID**	**integer**	**Unique number of node**
2	**EASTING**	**integer**	**Coordinate in x direction**
3	**NORTHING**	**integer**	**Coordinate in y direction**
4	ELEVATION	integer	Coordinate in z direction. Ignore
5	NOTES	string	Optional notes. Ignore

In consequence, a nodes file looks as follows:

```
ID<tab>EASTING<tab>NORTHING<tab>ELEVATION<tab>NOTES<ret>
1<tab>651700<tab>137200<tab>0<tab><ret>
2<tab>652220<tab>137600<tab>0<tab><ret>
...
```

The entries which are important for our do-it-yourself implementation are printed in boldface. Any information in the other columns will be ignored. That information may, however, be important to make other Transims modules work, most importantly the visualizer (Sec. 8). In particular, note the additional `<tab>` that separates a possibly empty `NOTES` field from the `<ret>`.

The link file has the following columns. Once more, the relevant ones are printed in bold; the other ones are just given for complete information.

Column	Header	Type	Explanation
1	**ID**	**integer**	**Unique ID number**
2	NAME	string	Name of the link, e.g. the street name. Ignore
3	**NODEA**	**integer**	**Node ID at one end of link**
4	NODEB	**integer**	**Node ID at other end of link**
5	PERMLANESA	integer	Number of lanes towards A. Ignore
6	PERMLANESB	integer	Number of lanes towards B. Ignore
7	LEFTPCKTSA	integer	Number of left pocket lanes towards A. Ignore
8	LEFTPCKTSB	integer	Number of left pocket lanes towards B. Ignore
9	RGHTPCKTSA	integer	Number of right pocket lanes towards A. Ignore
10	RGHTPCKTSB	integer	Number of right pocket lanes towards B. Ignore
11	TWOWAYTURN	boolean	Whether there is a two-way link for left turns in the middle of the road (an American specialty). Ignore
12	**LENGTH**	**positive float**	**Length of link in meters**
13	GRADE	float	Grade (= slope) of link. Ignore

14	SETBACKA	positive float	Setback distance (in meters) from the center of the intersection at node A. Ignore
15	SETBACKB	positive float	Setback distance (in meters) from the center of the intersection at node B. Ignore
16	CAPACITYA	positive float	Capacity of link towards A in vehicles per hour. Ignore (but see Sec. 18)
17	CAPACITYB	positive float	Capacity of link towards B in vehicles per hour. Ignore (but see Sec. 18)
18	SPEEDLMTA	positive float	Speed limit, in meters per second, towards A. Ignore (but see Secs. 17 and 18)
19	SPEEDLMTB	positive float	Speed limit, in meters per second, towards B. Ignore (but see Secs. 17 and 18)
20	FREESPDA	positive float	Free speed, in meters per second, towards A. Ignore (but see Secs. 17 and 18)
21	FREESPDB	positive float	Free speed, in meters per second, towards B. Ignore (but see Secs. 17 and 18)
22	FUNCTCLASS	keyword	Functional class of link. Ignore
23	THRUA	integer	ID of outgoing link across A which denotes “through” direction. Can be used for data compression. Ignore
24	THRUB	integer	ID of outgoing link across B which denotes “through” direction. Can be used for data compression. Ignore
25	COLOR	integer	Obsolete. Ignore
26	VEHICLE	keywords	Allowed modes on link. Ignore
27	NOTES	string	Arbitrary notes. Ignore

Task 6.1 *Generate a node file and a link file which together describe a square with a diagonal (i.e. four nodes and five links). You can use the files in*

`http://www.matsim.org/files/studies/test-net/network`

as a starting point.

6.3 Node class

```
typedef long Id;
typedef double Coord ;
...
class Node {
private:
    Id id_;
public:
    void set_id( Id val ) { id_ = val ; }
    Id   id() { return id_ ; }

private:
    Coord xx_;
public:
    void set_xx( Coord val ) { xx_ = val ; }
    xx() { return xx_ ; }

private:
    Coord yy_ ;
public:
```

```
    void set_yy( Coord val ) { yy_ = val ; }
    yy() { return yy_ ; }
};
```

6.4 SimWorld class

It is useful to have a `SimWorld` class that defines our simulation world:

```
class SimWorld {
public:
    typedef map<Id,Node*> Nodes ;
    Nodes nodes ;
    ...
    readNodes() ;
    ...
}
```

In this case, we will *not* make `Nodes` private, i.e. we will *not* encapsulate it. The result of this is that we can directly use the access functions of the STL. It is possible to use the STL functions even when `Nodes` is private, but we find the above solution easier for non-experts.

6.5 Nodes input

Reading the nodes file would go as follows:

```
#include <fstream>
#include <string>
...
const char* NODES_FILE_NAME = "T.nodes";
...
class Node {
    ...
};
...
class SimWorld {
    ...
};
...
void SimWorld::readNodes ( ) {
    cout << "\n### entering readNodes ...\n" ;
    ifstream inFile ; inFile.open(NODES_FILE_NAME) ;
    assert( inFile.is_open() ) ;
    string aString ;
    vector<string> tokens ;
    // process header line:
    getline( inFile, aString ) ;
    tokenize( aString, tokens ) ;
    assert( tokens[1]=="ID" ) ;
    assert( tokens[2]=="EASTING" ) ;
    assert( tokens[3]=="NORTHING" ) ;
    // main loop:
    while ( !inFile.eof() ) {
        getline( inFile, aString ) ;
        if ( !aString.empty() && isdigit( aString[0] ) )
                        // [[ skip lines with junk (e.g. last line) ]]
                {
            tokenize( aString, tokens ) ;
            Id nodeId ; convert( tokens[1], nodeId ) ;
            Coord xCoord ; convert( tokens[2], xCoord ) ;
            Coord yCoord ; convert( tokens[3], yCoord ) ;
            // initialize node:
            Node* node = new Node ;
            // enter node into node map:
            nodes[nodeId] = node ;
            node->set_id( nodeId ) ;
            node->set_xx(xCoord);
            node->set_yy(yCoord) ;
        }
    }
    cout << " nNodes: " << nodes.size() << endl ;
```

```
    cout << "### leaving readNodes ...\n\n" ;
}
```

The `convert` methods are as follows:

```
inline void convert ( const string& str, int& ii ) {
    ii= atoi( str.c_str() ) ;
}
inline void convert ( const string& str, long& ii ) {
    ii= atol( str.c_str() ) ;
}
inline void convert ( const string& str, double& dd ) {
    dd = atof( str.c_str() ) ;
}
```

This would be called from the main program via

```
int main()
{
        SimWorld simWorld ;
        simWorld.readNodes() ;
        ...
}
```

Task 6.2 *Write a program which reads the node data.*

6.6 Link class

The link class is analogous to the node class:

```
typedef double Len ;
typedef double Spd ;
...
class Link {
private:
    Id id_;
public:
    void set_id( Id val ) { id_ = val ; }
    Id   id() { return id_ ; }
private:
    Node* fromNode_;
public:
    void  set_fromNode( Node* node ) { fromNode_ = node ; }
    Node* fromNode() { return fromNode_ ; }
private:
    Node* toNode_ ;
public:
    void  set_toNode( Node* node ) { toNode_ = node ; }
    Node* toNode() { return toNode_ ; }
private:
    Len len_ ;
public:
    void set_length( Len val ) { len_ = val ; }
    Len  length() { return len_ ; }
};
```

6.7 Links input

Again, this is analogous to the nodes.

```
...
const char* LINKS_FILE_NAME = "T.links";
...
void SimWorld::readLinks ( ) {
    cout << "\n### entering readLinks ...\n" ;
    ifstream inFile ; inFile.open( LINKS_FILE_NAME ) ;
    string aString ;
    vector<string> tokens ;
    // process header line:
```

```
    getline( inFile, aString ) ;
    tokenize( aString, tokens ) ;
    assert( tokens[1]=="ID" ) ;
    assert( tokens[3]=="NODEA" ) ;
    assert( tokens[4]=="NODEB" ) ;
    assert( tokens[12]=="LENGTH" ) ;
    // main loop:
    while ( !inFile.eof() ) {
        getline( inFile, aString ) ;
        if ( !aString.empty() && isdigit( aString[0] ) ) {
            // ( skip lines w/ junk (e.g. last line) )
            tokenize( aString, tokens ) ;
            Id linkId ; convert( tokens[1], linkId ) ;
            Id fromNodeId ; convert( tokens[3], fromNodeId ) ;
            Id toNodeId ; convert( tokens[4], toNodeId ) ;
            Len length ; convert( tokens[12], length ) ;
            Link* link = new Link ;
            links[linkId] = link ;
            link->set_id ( linkId ) ;
            Node* fromNode = nodes[ fromNodeId ] ;
            assert( fromNode != NULL ) ;
            link->set_fromNode ( fromNode ) ;
            Node* toNode = nodes[ toNodeId ] ;
            assert( toNode != NULL ) ;
            link->set_toNode ( toNode ) ;
            link->set_length ( length ) ;
            fromNode->addOutLink(link) ;
            toNode->addInLink(link) ;
        }
    }
    cout << " nLinks: " << links.size() << endl ;
    cout << "### leaving readLinks ...\n\n" ;
}
```

Regarding `addOutLink` and `addInLink` see next section.

Task 6.3 *Write code that does the links input.*

Remember that you need to include `Links` into the `SimWorld` class similarly to `Nodes`.

6.8 Incoming/outgoing links

In order to traverse the graph, for each node we need the incoming and the outgoing links. Recall that for links we already have the corresponding information, i.e. the fromNodes and toNodes. The construction of the inLinks and outLinks is as follows:

First, add the corresponding entries to the node class:

```
class Node {
private:
    ...
    typedef vector<Link*> VLinks;
    Vlinks outLinks_;
    Vlinks inLinks_;
public:
    ...
    void addOutLink(Link* Link) { outLinks_.push_back(link); }
    Link* outLink(int i) { return outLinks_[i]; }
    int nOutLinks() { return outLinks_.size(); }

    void addInLink(Link* link) { inLinks_.push_back(link); }
    Link* inLink(int i) { return inLinks_[i]; }
    int nInLinks() { return inLinks_.size() ; }
} ;
```

Note that we do not need the associative array property here for `outLinks_` or `inLinks_`, and so we use the `vector` class instead of `map`.

Next, we generate the information of which links are incoming and outgoing. The easiest way is to add this in the `readLinks` routine at the end, as was already done in the previous section.

Task 6.4 *Add the information about incoming/outgoing links to your code.*

Task 6.5 *Test if you can read the network in*

```
http://www.matsim.org/files/studies/corridor/network
```

without errors.

Chapter 7

Cellular automata micro-simulation

7.1 Introduction

The micro-simulation executes the route plans and returns congestion levels. Since we do not have plans yet, we will at this stage see the traffic micro-simulation as something that moves vehicles along links and across intersections.

We use the same dynamics as we had used for the roundabout in Chap. 3. That is:

- The road is divided into cells of length 7.5 meters.
  We will only model links with single lanes.
- Each cell is either empty or occupied by exactly one vehicle.
- Vehicles move deterministically by one cell between time t and time $t+1$ if the cell ahead is empty at time t.
- Across intersections, we will check that the first cell of the receiving link is empty.

Implementation

7.2 Vehicles

Now, we need vehicles. We will start very simplistic:

```
class Veh {
private:
    Id id_ ;
public:
    set_id( Id val ) { id_ = val ; }
    Id id() { return id_ ; }
}
```

7.3 Vehicles on links

Now we need to extend the links so that they contain the vehicles. For our cellular automata (CA) approach, we represent the road by a 1-lane sequence of cells. In consequence,

```
class Link {
    ...
private:
    typedef vector<Veh*> Cells ;
    Cells cells_ ;
public:
    build() ;
}
```

As one sees, the road is a vector of pointers to `Veh`. If this pointer is `NULL`, then the corresponding cell is empty.

For modular programming, one would in fact introduce a new class, say `simlink`, and make it inherit from the `link` class. Unfortunately, this eventually means to templatize the link and node classes, which we do not want to do at this point. Further details are discussed in Chap. 10.

The `build()` command builds the road, i.e. reserves memory etc.:[1]

```
void Link::build () {
    int nCells ;
    nCells = int( length() / LCELL ) ;
    for( int ii=0; ii<nCells; ii++ ) {
        cells_.push_back(NULL);
    }
}
```

`LCELL` is a global constant containing the length of a cell which we set to 7.5 meters. According to the code, the number of cells is

$$N_{cells} = L/\ell, \tag{7.1}$$

where L is the length of the link and ℓ the length of a cell. `push_back` is the command to add elements to a `vector`.[2]

We also need functions to add vehicles at the upstream end and remove them at the downstream end of the link. Similarly, one needs to be able to test for the availability of space, and get access to the most downstream of the vehicles. The code segment looks as follows:

```
class Link {
    ...
    void addToLink( Veh* veh ) {
        assert( cells_[0]==NULL );
        cells_[0] = veh ;
    }
    veh* firstOnLink() {
        return cells_.back() ;
    }
    void rmFirstOnLink() {
        assert( cells_.back()!=NULL ) ;
        cells_.back() = NULL ;
    }
    bool hasSpace() {
        return cells_.front()==NULL ;
    }
}
```

`cells_.front()` and `cells_.back()` are STL functions and provide access to the first and the last element of the vector.

Finally, we need a method to move vehicles forward. This can look as follows:

```
class Link {
    ...
    void moveOnLink( int& nVehs ) ;
    void move( int& nVehs ) {
        moveOnLink( int& nVehs ) ;
        // more here to be added later ...
    }
} ;
```

[1] Again, there are specific commands in the STL to achieve the same thing. We leave that to the experts.

[2] One could use `allocate`, but the use of `push_back` preserves at least somewhat the look and feel of a traditional array.

and:

```
void Link::moveOnLink ( int& nVehs ) {
    int last = cells_.size() - 1 ;
    for( int ii=0; ii<last ; ii++ ) {
        Veh* veh = cells_[ii] ;
        if ( veh != NULL ) {
            nVehs ++ ;
            if ( cells_[ii+1] == NULL ) {
                cells_[ii+1] = veh ;
                cells_[ii] = NULL ;
                ii++ ;
                veh->set_speed( LCELL ) ;
            } else {
                veh->set_speed( 0. ) ;
            }
        }
    }
}
```

Note that this uses traditional array syntax, so alternative models can be easily implemented even by programmers not fluent in C++.

7.4 Random moves through intersections

We also need a method to move through intersections. If there is more than one outgoing link, then the vehicle needs to select one of those. In Sec. 9.1 we will introduce route plans for this purpose. In order to test the code without that functionality, here we introduce a method with random selection of the outgoing link:

```
class Node {
    ...
public:
    void rndmove() ;
    void move() {
        rndmove() ;
    }
}
```

and

```
void Node::rndmove ( ) {
    for ( VLinks::iterator ll=inLinks().begin(); ll!=inLinks().end(); ++ll ) {
        Link* inLink = (Link*) *ll ;
        Veh* veh = inLink->firstOnLink() ; // NULL if none
        if ( veh != NULL ) {
            int nOutLinks = outLinks().size() ;
            int outLinkIdx = int( myRand() * nOutLinks ) ;
            Link* theOutLink = outLink(outLinkIdx) ;
            if ( theOutLink->hasSpace() ) {
                inLink->rmFirstOnLink() ;
                theOutLink->addToLink( veh ) ;
            }
        }
    }
}
```

Note that in contrast to earlier no "`->second`" is used with the iterator, since the `VLinks` is a standard vector (array) structure, and not a `map`.

`myRand()` is a random number generator that returns values between zero (included) and one (excluded), for example

```
double myRand() {
    return rand()/(RAND_MAX+1) ;
}
```

7.5 Fairer intersections

In this text, an attempt is made to present a simple (the simplest?) version here, and to wait with improvements until Part III. In this section, there will be an exception: The modification presented here is not strictly necessary. Not including it does, however, result in strong artifacts and asymmetries in the traffic dynamics.

A disadvantage of the above code for intersection movement is that certain incoming links always get served earlier than others. A useful way to improve the situation is to go through the incoming links in random sequence. This can be achieved by

```
typedef multimap<double,Link*> RndLinks ;
RndLinks rnd_links ;
// go through all inLinks, give them a random number, and insert
// them according to it:
for ( VLinks::iterator ll=inLinks_.begin(); ll!=inLinks_.end();
      ++ll ) {
    Link* link = *ll ;
    rnd_links.insert( make_pair( myRand(), link ) ) ;
}
// retrieve the inLinks in the order of their random numbers:
for ( RndLinks::iterator ll = rnd_links.begin();
      ll != rnd_links.end(); ll++ ) {
    Link* inLink = ll->second ;
```

and then continue as above.

The above algorithm goes through all incoming links and gives them a random number and then inserts them into the multimap using the random number as key. A `multimap` is similar to the `map` we used for links and nodes with the only difference that keys do not have to be unique; this is necessary since it could happen that two random numbers are identical. The links are then taken out of the multimap in increasing order of the random number.

7.6 Initializing vehicles for testing purposes

We need to be able to put vehicles on the network. A useful method for this will be discussed in Chap. 9 in conjunction with the introduction of plans. Here we just point out that for testing purposes one can put vehicles on links for example as follows:

```
Id cnt = 0 ;
for ( Links::iterator ll=links.begin(); ll!=links.end(); ++ll ) {
    Link* link = ll->second ;
    Veh* veh = new Veh ;
    veh->set_id(cnt) ;
    cnt++ ;
    link->addVeh( Veh ) ;
}
```

7.7 Main program

Finally all the above functionality needs to be put together. This can be done as follows:

```
typedef double Time ;
...
Time globalTime = -1 ; // global definition of a time; see text
...
class Link ; // forward declaration
class Node {
    ...
};
class Link {
    ...
} ;
class Veh {
```

```
    ...
} ;
class SimWorld {
    ...
    void simulate() { // see later
        ...
    }
} ;
...
int main () {
    // network construction as discussed earlier
    ...

    // build the links:
    for ( SimWorld::Links::iterator ll =simWorld.links.begin();
                                      ll!=simWorld.links.end();
                                    ++ll ) {
        Link* link = ll->second ;
        link->build() ;
    }

    // insert some vehicles as explained above
    ...

    // time iteration:
    for ( globalTime=simStartTime; globalTime<99999; globalTime++ ) {
        bool done = false ;
        simWorld.simulate( done ) ;
        if ( done ) break ;
    }
    return 0;
}
```

and finally

```
void SimWorld::simulate ( bool& done ) {
    int nVehs=0 ;
    // links movement:
    for ( Links::iterator ll=links.begin(); ll!=links.end(); ++ll ) {
        Link* theLink = ll->second ;
        theLink->move( nVehs ) ;
    }
    // intersection movement:
    for ( Nodes::iterator nn=nodes.begin(); nn!=nodes.end(); ++nn ) {
        Node* theNode = nn->second ;
        theNode->move( ) ;
    }
    // output
    int skip=60 ;
    if ( long(globalTime)%skip==0 ) {
        for ( Links::iterator ll=links.begin(); ll!=links.end(); ++ll ) {
            Link* theLink = ll->second ;
            theLink->writeVehFile( ) ;
        }
    }
    if ( long(globalTime)%1000==0 ) {
        cout << "Step: " << globalTime
             << " NVehs: " << nVehs
             << endl ;
    }
    done = false ;
    if ( nVehs==0 ) {
        done = true ;
    }
}
```

The above code fragment also contains a provision for visualizer output, to be used in the next chapter.

Note the time is defined globally as `globalTime`. There are better ways to do this; this is, as always in this text, left to the experts.

Chapter 8

Visualizer

8.1 Introduction

For larger simulations, visualization is nearly always an absolute necessity. Writing a visualizer, however, goes beyond the purposes of this text. One option is the Transims visualizer, on which the output formats in the following are based; since the whole Transims package is available to academic institutions for an affordable license fee, this may be an option. In some cases, visualizers of other transportation simulation software may be available. In this section it will be described how a graphics program that plots data points based on Cartesian coordinates can be used to generate some basic visualization. The public doman software "gnuplot" will be used. Other plotting packages with similar functionality should also work.

Implementation

8.2 Vehicle output

The file format for vehicle output is as follows:

Column	Header	type	explanation
1	**`VEHICLE`**	**integer**	**Vehicle ID**
2	**`TIME`**	**integer**	**Current time (in seconds past midnight)**
3	**`LINK`**	**integer**	**Link ID**
4	**`NODE`**	**integer**	**FromNode ID (i.e. ID of node where the vehicle is coming from)**
5	**`LANE`**	**integer**	**Lane the vehicle is on**
6	**`DISTANCE`**	**float**	**Distance (in meters) the vehicle is away from the node**
7	`VELOCITY`	float	Vehicle speed (in meters per second)
8	`VEHTYPE`	integer	Vehicle type. "1" = car.
9	`ACCELER`	float	Vehicle acceleration (in m/s per second)
10	`DRIVER`	integer	Driver ID
11	`PASSENGERS`	integer	Number of passengers in vehicle
12	**`EASTING`**	**float**	**Position of vehicle in x direction**
13	**`NORTHING`**	**float**	**Position of vehicle in y direction**
14	`ELEVATION`	float	Position of vehicle in z direction
15	**`AZIMUTH`**	**float**	**Vehicle's orientation (degrees from east in counterclockwise direction)**

16	USER	integer	User-defined data field

The most important fields for our purposes here are time and the two spatial coordinates. When these fields are filled out correctly, the Transims visualizer will work even when all other fields are filled with dummy variables.

Some linear algebra is necessary to calculate the position and the orientation of the vehicles. It goes as follows:

1. The vector from the fromNode s to the toNode t is

$$\mathbf{r}_{st} = \begin{bmatrix} x_{st} \\ y_{st} \end{bmatrix} = \begin{bmatrix} x_t \\ y_t \end{bmatrix} - \begin{bmatrix} x_s \\ y_s \end{bmatrix} \tag{8.1}$$

2. When θ is the angle between the x axis and $\mathbf{r}$, then one has

$$\tan\theta = \frac{y}{x} \quad \text{or} \quad \theta = \begin{cases} \arctan\left(\frac{y}{x}\right) & \text{if } x > 0 \\ \arctan\left(\frac{y}{x} + \pi\right) & \text{if } x < 0 \\ \frac{1}{2}\pi & \text{if } x = 0 \text{ and } y > 0 \\ \frac{3}{2}\pi & \text{if } x = 0 \text{ and } y < 0 \end{cases} \tag{8.2}$$

3. A vehicle's distance on the link from the fromNode is given by the position of it's cell; if the cell number is i, then the position is $(i+1)\,\ell$, where ℓ is the length of a cell (typically 7.5 meters).
4. The coordinates of the vehicle now essentially are

$$\begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} x_s \\ y_s \end{bmatrix} + \begin{bmatrix} d\,\cos\theta \\ d\,\sin\theta \end{bmatrix} \tag{8.3}$$

5. After this calculation, vehicles are on the direct line between two nodes. What is missing is the offset depending on the lane the vehicle is in. This is just

$$\begin{bmatrix} +w\,\sin\theta \\ -w\,\cos\theta \end{bmatrix} , \tag{8.4}$$

which is added to Eq. (8.3). w is the width of a lane, for example 3.75 meters. Large values of w are often useful to "pull" road directions apart, which is useful when zooming out.

Corresponding code is

```
void Link::writeVehFile ( ) {
    static int first=1 ;
    static ofstream snapshotFile ;
    if ( first==1 ) {
        first = 0 ;
        snapshotFile.open( SNAP_FILE_NAME ) ;
        assert( snapshotFile.is_open() ) ;
        snapshotFile << "VEHICLE"
                     << '\t' << "TIME"
                     << '\t' << "LINK"
                     << '\t' << "NODE"
                     << '\t' << "LANE"
                     << '\t' << "DISTANCE"
                     << '\t' << "VELOCITY"
                     << '\t' << "VEHTYPE"
                     << '\t' << "ACCELER"
                     << '\t' << "DRIVER"
                     << '\t' << "PASSENGERS"
                     << '\t' << "EASTING"
                     << '\t' << "NORTHING"
                     << '\t' << "ELEVATION"
                     << '\t' << "AZIMUTH"
                     << '\t' << "USER"
                     << endl;
```

```
      }
      assert( snapshotFile.is_open() ) ;
      // write TWO empty lines between time steps:
      static Time lastTimeStep = -1 ;
      if ( lastTimeStep != globalTime ) {
          snapshotFile << "\n\n" << endl ;
          lastTimeStep = globalTime ;
      }
      // go through all cells of the link:
      for ( int ii=0; ii<cells_.size(); ii++ ) {
          // check if cells have a vehicle on them:
          if ( cells_[ii] != NULL ) {
              // get the veh and its position on the link:
              Veh* theVeh = cells_[ii] ;
              double pos = 7.5*(ii+1) ;
              int lane = 1 ;
              // calculate geographical coordinates and azimuth:
              Coord DX = - fromNode()->xx() + toNode()->xx() ;
              Coord DY = - fromNode()->yy() + toNode()->yy() ;
              typedef double Angle ;
              Angle theta = 0. ;
              if ( DX > 0 ) {
                  theta = atan( DY/DX ) ;
              } else if ( DX < 0 ) {
                  theta = PI + atan( DY/DX ) ;
              } else {
                  if ( DY > 0 ) { theta = PI/2. ; }
                  else { theta = - PI/2. ; }
              }
              if ( theta < 0. ) theta += 2.*PI ;
              double azimuth = theta/(2.*PI)*360 ;
              Coord easting = fromNode()->xx() + cos(theta) * pos
                  + sin(theta) * LANE_WIDTH * lane ;
              Coord northing = fromNode()->yy() + sin(theta) * pos
                  - cos(theta) * LANE_WIDTH * lane ;
              Coord elevation = 0. ;
              // write the information to the file:
              snapshotFile << theVeh->id()
                           << '\t' << globalTime
                           << '\t' << id() // link id
                           << '\t' << fromNode()->id()
                           << '\t' << lane
                           << '\t' << pos
                           << '\t' << theVeh->speed()
                           << '\t' << 1 // vehtype
                           << '\t' << 0. // acceleration
                           << '\t' << theVeh->id() // driver id
                           << '\t' << 0 // number of passengers
                           << '\t' << easting
                           << '\t' << northing
                           << '\t' << elevation
                           << '\t' << azimuth
                           << '\t' << 0 // user definable field
                           << "\n" ;
          }
      }
  }
```

For Transims, the header line is significant. For other systems, it may be omitted.

Note the two empty lines between time steps. The empty lines are important for the gnuplot visualization explained below; they are not important for the Transims visualizer and probably not for many other visualizers.

The above is called via

```
  void simulate (...) {
      ...
      for (Links::iterator ll = links.begin(); ll != links.end(); ll++ ) {
          Link* link = ll->second;
          link->writeVehFile(simTime) ;
      }
      ...
  }
```

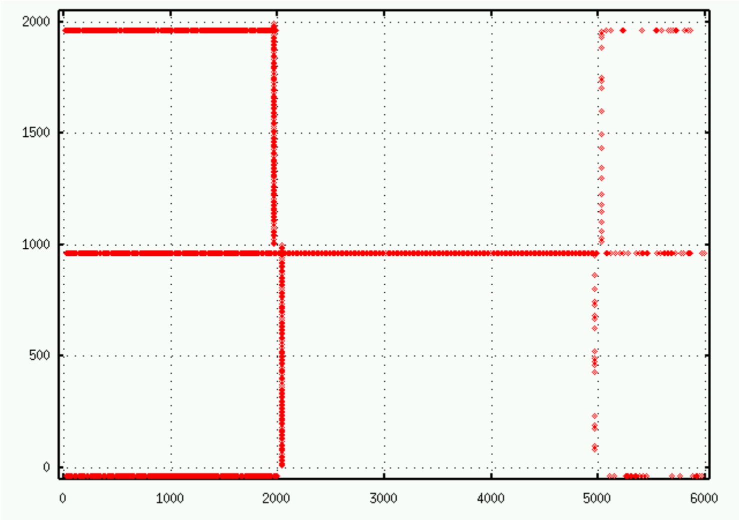


Figure 8.1: Vehicle snapshot using gnuplot.

8.3 Visualization via gnuplot

Gnuplot (`www.gnuplot.info`) is a plotting package that is available on most linux installations. In the following we will use it for a simple visualization of our traffic simulation results.

First, generate, in the same directory as where you have the vehicle snapshot file, a file named `gpl` with the following contents:

```
a=a+1
set grid
set xrange[-50:6050]
set yrange[-50:2050]
print a
plot "T.veh" index a u 12:13 t ""
if ( a < 200 ) reread
a = 0
```

This assumes that your vehicle snapshot file is called `T.veh`.

Start gnuplot by typing `gnuplot`. Inside gnuplot, type

```
gnuplot> a=1
gnuplot> load 'gpl
```

The result should be a window similar to Fig. 8.1 displaying the status of the simulation time step by time step.

8.4 Testing the current status of the simulation

Task 8.1 *Before one continues, one should make some tests if the simulation really works. Build a square* ***with a diagonal****. As suggested before: Just start from*

`http://www.matsim.org/files/test-net/network .`

Try the following things, and check them with the visualizer:

1. *For initialization, completely fill one of the links with vehicles. Do they move the way you would expect? What would you expect? Are all links used? Remember that the link decision on intersections is random at the moment.*

2. *For initialization, completely fill the two links* ***which go into the same node*** *with vehicles. What happens at the merge? Who has the priority in your code? Why?*

Chapter 9

Plans following in the micro-simulation

9.1 Plans

In our micro-simulation, travelers follow plans. In our do-it-yourself traffic simulation, we only look at cars. Cars have complete routes in their plans.

Route plans always include variants of the following information:

StartTime, StartLoc, Node1, Node2, ..., EndLoc.

- StartTime: Time-of-day when the traveler wants to start. We always use seconds past midnight.
- StartLoc: Starting location. For us, this is the link ID where the trip starts.
- Node1: First node of route plan.
- Node2, etc.: The following nodes of the route plan.
- EndLoc: The final destination of the trip. For us, this is the link ID where the trip ends.

In terms of programming, this means:

1. We need a mechanism to read plans.
2. We need a data structure ("parking queue") where to keep vehicles/plans until their starting time.
3. We need a data structure ("waiting queue") where to keep vehicles/plans which are beyond their starting time, but have not been able to move into the traffic because of congestion.
4. We need a mechanism to move vehicles from the parking queue to the waiting queue.
5. We need a mechanism to move vehicles from the waiting queue on to the start link.
6. We need a mechanism to move vehicles across an intersection so that they follow plans.

In principle, the plans file can contain the whole daily plan for each individual traveler in the simulation. For the time being, we will however identify car trips and vehicles, and skip the remaining information in the plans file, if any.

Implementation

9.2 Vehicle class

First we need to extend the vehicle class. An implementation is

```
#include <deque>
...
class Veh {
private:
    Id id_;
public:
    void set_id( Id val ) { id_ = val ; }
    Id id() { return id_ ; }
private:
    Spd speed_ ;
public:
    void set_speed( Spd tmp ) { speed_ = tmp ; }
    Spd speed() { return speed_ ; }
private:
    Time startTime_;
public:
    void set_startTime ( Time val ) { startTime_ = val ; }
    Time startTime() { return startTime_; }
private:
    Id arrivalLinkId_ ;
public:
    void set_arrivalLinkId( Id val ) { arrivalLinkId_ = val ; }
    Id arrivalLinkId() { return arrivalLinkId_ ; }
private:
    typedef deque<Id> Route;
    Route route_ ;
public:
    void addNodeId2Route(Id nodeId) { route_.push_back(nodeId); }
    Id nextNodeID() {
        if ( route_.size() >= 1 ) {
            return route_.front() ;
        } else {
            return -1 ;
        }
    }
    void incPlan() { route_.pop_front(); }
    void writeEvent(Id linkId, Id fNodeId, int flag) ;
    void dump() {
        cout << " vehid: " << id()
             << " speed: " << speed()
             << endl ;
    }
};
```

The `writeEvent` method will be explained later.

Note how the route plan is implemented as a deque, which is a data structure which makes it easy to add and remove elements at both ends.

9.3 Plans format

We use the Transims route format in order to have a well-defined standard.

For people who insist on their own format, it is in theory possible to write converters. In practice, this is nearly always a headache, since, for example: the converters are not maintained; third parties do not know where the executables are located or how they are used; plans files are huge (typically several GB) and for that reason one does not want different representations of the same information on the hard disk.

Clearly, a better choice for what we do would be XML (eXtended Markup Language). This is discussed in Sec. 24.3. The only disadvantage of XML is that one needs libraries (such as expat) for parsing, which means that our code would no longer be standalone. For that reason, for the time being we use the Transims format.

Transims organizes trips into legs, for example: walk to car, drive to office parking, walk to office. More precisely, a "trip" goes from one activity to the next, and legs are characterized by different modes of transportation. For our project here, we only look at car legs.

A typical example looks as this:

```
1 0 1 1 0 0
27825 100 2 1900 2
0 86400 0
1 0 1
8
1 0 40 70 100 130 160 190
```

Since the number of nodes varies from plan to plan, plans need to have a variable length part. In Transims this is achieved via a fixed length and a variable length part. The last token of the fixed length part says how many more tokens are to follow. The meaning of the individual numbers is as follows:

Fixed length part:

Number	explanation
1	**Traveler (Person) ID**
2	User field. Irrelevant for us
3	Trip ID. Irrelevant for us
4	Leg ID. Irrelevant for us
5	FirstLegFlag. Irrelevant for us
6	LastLegFlag. Irrelevant for us
7	**StartTime**
8	**StartLocation. = StartLink for us**
9	Type of StartLocation. Irrelevant for us
10	EndLocation. Irrelevant for us
11	Type of EndLocation. Irrelevant for us
12	Duration. Irrelevant for us
13	Stop Time. Irrelevant for us
14	MaxTimeFlag. Irrelevant for us
15	Driver Flag. Irrelevant for us
16	Mode. Should always be 0
17	Vehicle Type. Irrelevant for us
18	**Number of additional tokens (variable length part)**

The 7th token is the StartTime; the 8th token the StartLocation (which is, for us, the link on which the vehicle starts).

An important information is the 16th token of a block/leg, which codes the mode of transportation: "0" means "car". If, for a given block, one finds a different number here, we will ignore the whole block/leg and continue with the following one.

The 18th token of a block gives the number of the tokens following from there on.

Variable length part:

number	explanation
1	Vehicle ID. Ignore
2	Number of Passengers. Needs to be zero (because the meaning of the following data depends on this).
3	**Node 1**

4	Node 2
5	etc.

The 20th token (= 2nd token of variable length part) should be zero; if not, the plan should be skipped.[1]

All following tokens are NodeIDs. The first NodeID after the start link is included; as long as one uses uni-directional links (as we do), this information is redundant.

The full Transims route plans specification is in the Transims documentation:

`http://www.matsim.org/files/doc/transims-1.0/files.pdf`

Important: There are differences between the transims-1.0 plans format and the transims-1.1 plans format. We use the transims-1.0 plans format.

Important: Line breaks in the route plans are not significant. However, empty lines between blocks are significant. Each block corresponds to a leg.

Task 9.1 *Write a route plans file with exactly one route for "test-net".*

9.4 ReadPlans

Here is an example of how to read plans into the simulation:

```
void SimWorld::readPlans (Time& simStartTime ) {
    cout << "\n### entering readPlans ...\n" ;
    int cnt=0 ;
    Plan plan ;
    simStartTime=99999 ;
    while ( plan.readNextPlan()==0 ) {
        if ( plan.mode()!=0 ) {
            cout << " Wrong mode, skipping plan.\n" ;
        } else if ( plan.nPassengers()!= 0) {
            cout << " Wrong number of passngers; skipping plan.\n" ;
        } else {
            cnt++ ; if ( cnt%1000==0 ) { cout << " Cnt: " << cnt << endl ; }
            if ( plan.startTime() < simStartTime ) simStartTime = plan.startTime() ;
            Veh* veh = new Veh ;
            veh->set_id( plan.travId() ) ;
            veh->set_startTime( plan.startTime() ) ;
                        veh->set_arrivalLinkId( plan.endLinkId() ) ;
            assert( links[plan.startLinkId()]!=NULL ) ;
            links[plan.startLinkId()]->addToPark(veh) ;
            for ( int ii=plan.firstNodeIndex(); ii<=plan.lastNodeIndex(); ii++ ) {
                veh->addNodeId2Route( plan.nodeTokens(ii) ) ;
            }
        }
    }
    cout << " nPlans: " << cnt
         << " simStartTime: " << simStartTime
         << endl ;
    cout << "### leaving readPlans ...\n\n" ;
}
```

Notes:

- This also calls the vehicle initialization, and puts the vehicle into the waiting queue of the starting link. ***Remove the temporary way in which we had initialized vehicles earlier.***
- It also checks which is the earliest vehicle start time.

Since parsing the plans is a bit messy, parsing is delegated to a subroutine `readNextPlan`.

[1] If this token is not zero, then the following numbers are not only NodeIDs, but also passenger IDs. We do not want to treat this case.

```
int Plan::readNextPlan ( ) {
    static ifstream inFile;
    // open file if necessary:
    static int first=1 ; if ( first ) {
        first = 0 ;
        inFile.open(PLANS_FILE_NAME);
    }
    // always check if file is really open:
    assert( inFile.is_open() ) ;
    // main loop:
    while (!inFile.eof()) {
        // deal with junk:
        string line ; char ch = inFile.peek() ;
        if ( !isdigit(ch) ) {
            getline( inFile, line ) ;
        }
        // here is the real reading:
        else {
            // read fixed length part:
            for ( int ii=1; ii<=18; ii++ ) {
                inFile >> fixTokens_[ii] ;
            }
            // read variable length part:
            for ( int ii=1; ii<=fixTokens_[18]; ii++ ) {
                assert( ii <= MAXTOK_ ) ;
                inFile >> varTokens_[ii] ;
            }
            return 0 ;
        }
    }
    return 1 ;
}
```

9.5 Class Plan

A class `plan` is used to transmit the variables, which avoids an overly long argument list in the call to `ReadNextPlan`. This class specification also does the translation from numbered tokens to meaningful variables. The following also contains functions to set variables, which is not necessary for the purposes of this chapter. It will however become necessary in Chap. 11.

```
class Plan {
private:
    int fixTokens_[19] ;
    static const int MAXTOK_=2000 ;
    int varTokens_[MAXTOK_+1] ;
    static const int firstNodeIndex_ = 1 ;
    // (``const'' makes sure this cannot be changed; ``static'' is
    // necessary here because of the ``const''.)
public:
    Id travId() { return fixTokens_[1] ; }
    void set_travId( Id tmp ) { fixTokens_[1] = tmp ; }
    // --------------------------------------------
    Time startTime() { return fixTokens_[7] ; }
    void set_startTime ( Time tmp ) { fixTokens_[7] = int(tmp) ; }
    // --------------------------------------------
    Id startLinkId() { return fixTokens_[8] ; }
    void set_startLinkId( Id tmp ) { fixTokens_[8] = tmp ; }
    // --------------------------------------------
    Id endLinkId( ) { return fixTokens_[10] ; }
    void set_endLinkId( Id tmp ) { fixTokens_[10] = tmp ; }
    // --------------------------------------------
    int mode() { return fixTokens_[16] ; }
    int nPassengers() { return varTokens_[2] ; }
    // --------------------------------------------
    // vtok1 vtok2 vtok3 vtok4 vtok5 vtok6 ... vtok(L-2) vtok(L-1) vtok(L)
    //             node1 node2 node3 node4 ... node(N-2) node(N-1) node(N)
    // L = fixTokens_[18]
    // N = lastNodeIndex ;
    void set_nNodes( int tmp ) { fixTokens_[18] = tmp+2 ; }
    int nNodes() { return fixTokens_[18] - 2 ; }
```

```
    // ---------------------------------------------
    int firstNodeIndex() { return firstNodeIndex_ ; }
    int lastNodeIndex() { return firstNodeIndex_+nNodes()-1 ; }
    // ---------------------------------------------
protected:
    int tokIdx( int ii ) {
        return ii+3-firstNodeIndex_ ;
        // ( 1 + 3 - 1 = 3, where we find the first node )
        // ( N + 3 - 1 = N+2, where we find the last node )
    }
    // ---------------------------------------------
public:
    Id nodeTokens(int ii) {
        int index = tokIdx(ii) ;
        assert( index <= MAXTOK_ ) ;
        return varTokens_[index] ;
    }
    void set_nodeTokens( int ii, Id tmp ) {
        assert( ii >= firstNodeIndex() ) ;
        assert( ii <= lastNodeIndex() ) ;
        int index = tokIdx(ii) ;
        assert( index <= MAXTOK_ ) ;
        varTokens_[index] = tmp ;
    }
    int readNextTrip() ;
    int readNextPlan() ;
    int writePlan() ;
    void dump() ;
// constructor
    Plan() {
        for ( int ii=0; ii<=18; ii++ ) fixTokens_[ii]=0 ;
        fixTokens_[9] = 2 ; // StartLoc type = parking
        fixTokens_[11] = 2 ; // EndLoc type = parking
        fixTokens_[15] = 1 ; // traveler is driving
        fixTokens_[17] = 1 ; // vehicle type = auto
    }
} ;
```

9.6 Park queue

The park queue, as explained above, contains vehicles whose starting time is in the future. Here is a mechanism for the park queue.

```
class Link {
    ...
private:
    typedef multimap<Time,Veh*> ParkQueue ;
    ParkQueue parkQueue_ ;
public:
    void addToPark( Veh* veh ) {
        parkQueue_.insert( make_pair( veh->startTime(), veh ) ) ; // see txt
    }
    Veh* firstInPark() {
        if ( parkQueue_.size()>=1 ) {
            return parkQueue_.begin()->second ;
        } else {
            return NULL ;
        }
    }
    void rmFirstInPark() {
        assert( parkQueue_.size() >= 1 ) ;
        parkQueue_.erase( parkQueue_.begin() ) ;
    }
    ...
};
```

Note that the implementation for `ParkQueue` is

```
typedef multimap<Time,Veh*> ParkQueue ;
```

We have in fact already used a `multimap` for the implementation of “fair” intersections (Sec. 7.5). An additional function now is `erase()`.

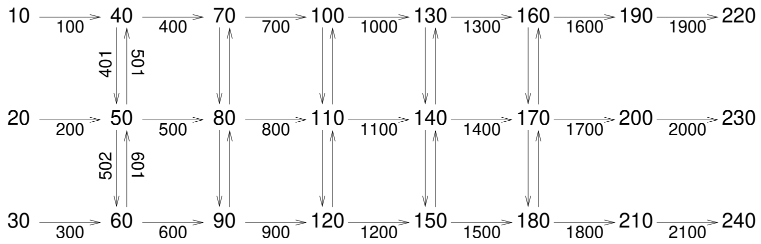


Figure 9.1: Sketch of the "corridor" network. The numbers give the corresponding node and link IDs.

Overall, this implements a priority queue, where the element with the lowest key is always available via `begin()`. "Lowest key" here means the earliest starting time.

9.7 Wait queue

The wait queue, as also explained above, contains vehicles whose starting time has passed but they have not made it into the traffic because of congestion. The separation between park and wait queue seems somewhat arbitrary at this point. It is necessary to provide an efficient way to write "events" when vehicles intend to start, even if they do not make it into the traffic in the same time step (Sec. 9.11).

Here is a mechanism for the wait queue:

```
class Link {
    ...
private:
    typedef deque<Veh*> WaitQueue ;
    WaitQueue waitQueue_ ;
public:
    void addToWait( Veh* veh ) {
        waitQueue_.push_back( veh ) ;
    }
    Veh* firstInWait() {
        if ( waitQueue_.size()>=1 ) {
            return waitQueue_.front() ;
        } else {
            return NULL ;
        }
    }
    void rmFirstInWait() {
        assert( waitQueue_.size() >= 1 ) ;
        waitQueue_.pop_front() ;
    }
    ...
};
```

Task 9.2 *Read your plans into your simulation.*

Task 9.3 *Read the network and the plans from*

`http://www.matsim.org/files/studies/corridor/teach`

into your simulation.

A sketch of the "corridor" network is given in Fig. 9.1.

9.8 Vehicle insertion

Vehicles need to be moved from the waiting queue into the traffic. We do this by

- moving the `SimLink::move(..)` function to `SimLink::moveOnLink(..)`, and then
- defining a new `SimLink::move(..)` function as follows:

```
class SimLink : public Link {
    ...
    void move ( int& nVehs ) {
        parkToWait() ;
        waitToLink() ;
        moveOnLink( nVehs ) ;
    }
} ;
```

The corresponding code is

```
void Link::parkToWait () {
    Veh* veh = firstInPark() ;
    while ( veh != NULL && veh->startTime() <= globalTime ) {
        rmFirstInPark() ;
        addToWait( veh ) ;
        Id linkId = id() ;
        Id fromNodeId = fromNode()->id() ;
        veh->writeEvent( linkId, fromNodeId, DEPARTURE_FLAG ) ;
        veh = firstInPark() ;
    }
}
```

and

```
void Link::waitToLink () {
    Veh* veh = firstInWait() ;
    while ( hasSpace() && veh != NULL ) {
        rmFirstInWait() ;
        addToLink( veh ) ;
        veh->incPlan() ; // easy to forget!!
        Id linkId = id() ;
        Id fromNodeId = fromNode()->id() ;
        veh->writeEvent( linkId, fromNodeId, WAIT_TO_LINK_FLAG ) ;
        veh = firstInWait() ;
    }
}
```

Overall, what we actually do is the following:

- During the initialization of the simulation, we read *all* the plans into computer memory. During this reading process, we also sort them by starting time into the parking queue.
- During the simulation itself, in each time step and for each link we check if the first vehicle in the parking queue is "due" for its entry into the traffic. If the answer is yes, then the vehicle is moved to the waiting queue. This is repeated until no more vehicles want to depart on this link in this time step.
- For all vehicles in the park queue, it is attempted to insert them into the traffic.

The meaning of `writeEvent` will be explained later.

9.9 Plans following and vehicle arrival

During the traffic simulation, the turning direction corresponding to the route plan needs to be found. That is, the random turning dynamics of Sec. 7.4 needs to be replaced by something like

```
void Node::move ( ) {
    // generate random sequence of inlinks as discussed earlier:
    typedef multimap<double,Link*> RndLinks ;
    RndLinks rndLinks ;
    for ( VLinks::iterator ll=inLinks().begin(); ll!=inLinks().end(); ++ll ) {
        Link* theLink = *ll ;
        double rnd = myRand() ;
```

```
            rndLinks.insert( make_pair( rnd, theLink ) ) ;
        }
        // go through that rnd sequence of inlinks and move vehicles
        // across intersection if possible:
        for ( RndLinks::iterator ll=rndLinks.begin(); ll!=rndLinks.end(); ll++ ) {
            Link* inLink = ll->second ;
            Veh* veh = inLink->firstOnLink() ; // NULL if none
            if ( veh != NULL ) {
                Id nextNodeId = veh->nextNodeID() ;
                if ( nextNodeId>0 ) {
                    Link* theOutLink = findOutLink( nextNodeId ) ;
                    if ( theOutLink->hasSpace() ) {
                        inLink->rmFirstOnLink() ;
                        theOutLink->addToLink( veh ) ;
                        veh->incPlan() ;
                    }
                } else { // end of plan
                    inLink->rmFirstOnLink() ;
                    Id arrivalLinkId = veh->arrivalLinkId() ;
                    // WARNING: one should check if the arrivalLink is
                    // connected to the current node!!
                    veh->writeEvent( arrivalLinkId, inLink->toNode()->id(), ARRIVAL_FLAG ) ;
                    delete veh ;
                }
            }
        }
    }
```

Note that the event uses the id of the arrival link, not the current link id.

Task 9.4 *Run your simulation with the network from*

```
http://www.matsim.org/files/studies/corridor/network
```

and plans from

```
http://www.matsim.org/files/studies/corridor/teach/0.plans
```

Results should be submitted as T.veh and T.bin files taken ***every 60 seconds****.*
When does the last vehicle leave your simulation? (Answering this question is important since it allows us to compare results.)

9.10 Computational Speed

Since in the application, many of the problems are fairly large, one needs to keep an eye on computing speed. A useful measure for this are "vehicle updates per second". Let's say that for a simulation with 10^4 vehicles and 10^3 time steps we need 10 seconds of computing time. Then we have $10^4 \times 10^3 = 10^7$ vehicle updates per 10 seconds, or 10^6 vehicle updates per second. This number is typical for a simple implementation on a 300 MHz CPU.

Under unix one obtains the computing speed for example via `time` (see man-page). My personal result looks like

```
92.88user 0.00system 1:34.50elapsed 98%CPU (0avg...
```

We are most interested in "92.88user" (coresponding to 92.88 sec).

Transportation science sometimes does the "real time limit" (for our purposes $=$ the number of vehicles with which the simulation runs as fast as reality).

All of these values depend on the vehicle density, which therefore always needs to be given when giving computing speeds.

Task 9.5 *How long does your simulation for the "corridor" network with 0.plans take to run? Please also tell us your implementation (C++ or Java or ??). Do this once with output and once with output switched off. What does this roughly correspond to in "vehicle updates per second". How did you obtain that number?*

9.11 Events output

Besides visualizer output, we need some output that is geared more towards the internal functionality of the system. We call this “events output”. The name means that events output is triggered by some event. Typical events are vehicle departure, vehicle arrival, or link traversal.

Specifically, our events file consists of the following fields. From now on, we deviate from Transims formats and use our own formats. The main reason is that the remaining files are not very large and thus converting them when necessary seems justified. As argued elsewhere, in the longer run these files should all be in XML format.

Column	Header	type	explanation
1	**TIMESTEP**	**int**	**time step**
2	**VEHICLEID**	**int**	**vehicle id**
3	**LINK**	**int**	**Link ID**
4	FROMNODE	int	FromNode ID for link. Irrelevant for us since we use uni-directional links
5	**FLAG**	**int**	**0: vehicle arrives at final destination** **2: vehicle leaves a link to go across an intersection** **4: vehicle moves from wait queue into traffic** **5: vehicle enters a link coming from an intersection** **6: vehicle is supposed to start**
6	NOTES	string	notes (leave empty, but separate by tab)

These events will be needed later when we introduce feedback and learning.

Task 9.6 *Write code which writes all of the above events to file when they are encountered.*

Chapter 10

Modularization, inheritance, templates, and code re-use

10.1 Introduction

As discussed in Chap. 2, transportation simulation packages consist of many modules. So far, we have seen the traffic simulation and the visualizer. The next module will be the router.

In contrast to the visualizer, our router will operate on a graph similar to the traffic simulation. This means that it makes sense to re-use some of the traffic simulation code. There are several options:

- If your are working as part of a team and your task is the router, then you can just delete the pieces of code that are specific to the traffic simulation (example: the cell structure of the links) and go from there.
- If you want one consistent piece of code but not many hassles in terms of software design, then one option is to have the functionality for the simulation and for the router combined in the same software. A link for example would keep the cell structure, even when used by the router.

  This is quite inefficient both in terms of performance and in terms of memory usage, but our experience is that for the examples discussed in this text this is a workable solution. In this case, you do not need to read this chapter.
- It is possible to separate the general purpose pieces of the network reading and network construction from the simulation specific pieces.

It is the last point that will be discussed in this chapter.

10.2 Links, Simlinks, and Inheritance

It makes sense to separate the graph functionality that will be used by several modules from the graph functionality that is used by a single module only. The mechanism to do this is inheritance. For example

```
class Link {
private:
    Id id_ ;
public:
```

```
    void set_id( Id val ) { id_ = val ; }
    Id   id() { return id_ ; }
private:
    Len len_ ;
public:
    void set_length( Len val ) { len_ = val ; }
    Len  length() { return len_ ;}
    ...
} ;

class SimLink : public Link {
private:
    Cells cells_ ;
public:
    void build() ;
    void addVehToLink( Veh* veh ) ;
    ....
}
```

This means that SimLink can do everything that Link can do, plus additional things. For example:

```
...
Link* link ;
SimLink* simLink ;
...
cout << link->id() ; // o.k.
cout << simLink->id() ; // o.k., simlink is a link
link->build() ; // not o.k., link is not a simlink
simLink->build() ; // o.k.
```

The word `public` in `class SimLink : public Link` means that everything that was public in `Link` will be available for `SimLink`. For the purposes of these things, `SimLink` will behave exactly as `Link`.

This is the only type of inheritance that we will consider.

10.3 Templates

Inheritance, without additional measures, does not work for graph reading and graph construction. It is not possible to do something like

```
class Node ; // forward declaration
class Link {
    ...
    Node* toNode() { return toNode_ ; }
    ...
} ;
...
class SimLink : Link {
    ...
} ;
...
int main () {
    ...
    SimLink* aSimLink = new SimLink( ... ) ;
    ...
    SimNode* aSimNode = aSimLink->toNode() ;  // does not work
}
```

because `toNode()` is of type `Node*` instead of of type `SimNode*`.

For C programmers and many other people, it will be clear that it is possible to work around this problem: this is just about pointers, and it should be possible to cast pointers to whatever one wants. In general, it is however an advantage that C++ enforces consistency between pointer objects, and so one should not deliberately circumvent this type checking.

A possibility to work around this is the use of templates.

```
template <class Node>    // <======
class Link {
    ...
    Node* toNode() { return toNode_ ; }
    ...
} ;
...
class SimNode ; // forward declaration
class SimLink : Link<SimNode> {   // <======
    ...
} ;
...
int main () {
    ...
    SimLink* aSimLink = new SimLink( ... ) ;
    ...
    SimNode* aSimNode = aSimLink->toNode() ;   // works
}
```

In fact, not much seems to have changed. What is the difference?

Template classes are often described as "parameterized classes". In fact, one could have written

```
template <class XXnode>    // <======
class Link {
    ...
    XXNode* toNode() { return toNode_ ; }
    ...
} ;
```

where now the notation `XXnode` makes clear that the type of the node is left open.

Then, when later saying

```
class SimNode ;
class SimLink : Link<SimNode> {
    ...
} ;
```

then this means that `SimLink` inherits from `Link` while using `SimNode` everywhere where `XXnode` is in the definition. In consequence,

```
aSimLink->toNode() ;
```

now returns a pointer to `SimNode`.

Thus, a method to translate everything we have done so far into a more general network construction is to write things like

```
// --------------------------------------------
template <class Node>
class Link {
    ...
} ;
template <class Link>
class Node {
    ...
} ;
template <class Node,class Link>
class Net {
public:
    typedef map<Id,Node*> Nodes ;
    Nodes nodes ;
    ...
    void readNodes() {...} ;
    ...
} ;
// --------------------------------------------
class SimNode ; // forward declaration
class SimLink : public Link<SimNode> {
    ...
} ;
class SimNode : public Node< SimLink> {
    ...
} ;
```

```
class SimWorld : public Net<SimNode,SimLink> {
    ...
} ;
// --------------------------------------------
int main () {
    ...
    SimWorld simWorld ;
    ...
    simWorld.readNodes() ;
    simWorld.readLinks() ;
    ...
}
// --------------------------------------------
```

In spite of the above explanation, for an inexperienced programmer the above is probably too much of a change to be done in one step and it will be necessary to achieve some familiarity with templates based on simpler programs before achieving this task. We hope that the above notes can guide the necessary reading and experimentation when templatization of the transportation simulation is the goal.

10.4 What belongs into the base class?

It is never simple to decide what belongs at what level of the hierarchy in inheritance. A possibility is to have only the basic things for graph construction in the base class and everything else in the derived class. This would mean to have ID, toNode, fromNode, and possibly inLinks and outLinks in the base class and everything else in the derived classes.

We do however think that it makes more sense to have everything that is in the nodes and links data files in the base class. In that way, the programs for reading the network data can be used by all modules without any changes, and the memory overhead is still not too bad.

Chapter 11

Route planner

11.1 Introduction

In Chap. 9 we have modified the traffic simulation in a way that each individual vehicle follows precomputed plans. In this Chapter, we will discuss a simple method to generate these route plans. For the sake of simplicity, we continue to only look at the car mode, which describes 80 percent or more of all travel in most western cities. Routing for other modes will be discussed in Sec. 20.

For each traveler, the input to the router consists of the following information:

- Trip Start Time.
- Trip Start Location. LinkID where the trip starts.
- Trip End Location. LinkID where the trip ends.

The output is a plans file, as specified in the previous section.

11.2 Fastest Path

The typical method to obtain routes is to calculate fastest paths. This is achieved via a standard shortest path algorithm by using link travel time as link cost. These algorithms typically go from node to node, which means that we have to translate our starting and ending locations to the corresponding nodes. Such an algorithm (Dijkstra algorithm, see e.g. ?) then can proceed as follows:

- Set `arrTime` at all nodes to infinity. Set `isDone` of all nodes to `false`.
- Take the starting node from the trip. Make it the current node. Set its `arrTime` to the trip starting time.
- "Node expansion:" Set `isDone` of the current node to `true`. Go through all outgoing links from the current node. For each such link, calculate arrival time at `toNode` as

  $$\texttt{tmpArrivalTime} = \texttt{now} + \texttt{outLinkTravelTime}\,, \tag{11.1}$$

  where `now` is the `arrTime` at the current node.

  If `tmpArrivalTime` is smaller than `toNode`'s current `arrTime`, then a faster path to that node just has been found. In that case,

  – Set `toNode`'s `arrTime` time to `tmpArrivalTime`.
  – Set a pointer at `toNode` pointing back to the current node.

- Out of all nodes where `isDone` is false, take the one with the minimum `arrTime`. Do "node expansion" with this node.
- Etc.

One can stop when the destination node is about to be expanded. ***Note that one cannot stop when the end node is touched for the first time (i.e. when its time is set from infinity to some finite value) since some better time can be found later.*** The full path can now be found by taking the end node, and following the pointers back to the start node.

11.3 Link travel times

What is missing is the value of `outLinkTravelTime`. When no other information is available, then we use

$$\text{linkTravelTime} = \text{linkLength}/\text{linkFreeSpeed}\ . \tag{11.2}$$

For the CA traffic simulation, the free speed is one cell per time step, or 7.5 m/s.

Congestion will reduce the speeds on the links. This effect is included into the router in Chap. 12.

Implementation

11.4 Library support for graph algorithms

There are libraries for graph algorithms, such as LEDA. In the past, they were never flexible enough to cover everything we want to do (e.g. time dependence). This will eventually change, and there will be options to pass calls to arbitrary cost functions to a graph algorithm. Once that works, writing router code will become considerably simpler.

11.5 General structure

The general structure of the router is as follows (not assuming the use of templates as discussed in Chap. 10):

```
class Link ;
class Node {
    ...
};
class Link {
    ...
};
class Plan {
    ...
} ;
class RouteWorld {
private:
    typedef map<Id,Node*> Nodes ;
    Nodes nodes ;
    typedef map<Id,Link*> Links ;
    Links links ;
public:
    void findPath( Plan& ) ;
} ;
```

```
...
int main() {
    // instantiate routeWorld:
    RouteWorld routeWorld ;

    // read the network:
    routeWorld.readNodes() ;
    routeWorld.readLinks() ;

    // main loop:
    Plan plan ;
    while ( plan.readNextTrip()==0 ) {
        routeWorld.findPath( plan ) ;
        plan.writePlan() ;
    }
}
```

As discussed in Chap. 10, the node, link, and plan classes and methods can be taken from previous chapters. Depending on the intention, one can just copy them into the route code and comment out unneeded portions. Alternatively, one can put them into a separate file and include them both into the simulation and into the router code. As discussed in Chap. 10, the best solution would be to use inheritance, which however implies the use of templates.

11.6 Input file: Trips

Transims does not have a trips file; indeed, the same information can be derived from Transims activity files (see Sec. ??). Transims activity files contain much more information than we need here, and they have been a continuous source of error and misunderstanding. And as a final argument, we believe that the activities file should be an XML subset of the plans file, as we will discuss in Sec. 24.3. For all those reasons, at this point we deviate once more from Transims file formats and introduce our own file format for trips.

The format is as follows:

Column	Header	type	explanation
1	**ID**	**integer**	**ID number of traveller/vehicle**
2	DEPTLINK	**integer**	**departure location (link ID)**
3	**ARRLINK**	**integer**	**arrival location (link ID)**
4	**TIME**	**integer**	**departure time of traveller/vehicle in "seconds past midnight"**
5	NOTES	string	notes (leave empty, but separate by tab)

This can be read in a similar way as a links or nodes file; and we will use the already existing `plan` class for storing the information. In consequence, reading the trips looks as follows:

```
int Plan::readNextTrip () {
    static ifstream inFile ;
    string aString ;
    vector<string> tokens ;
    static bool first=true ; if ( first ) {
        first = false ;
        // open file:
        inFile.open( TRIPS_FILE_NAME ) ;
        assert( inFile.is_open() ) ;
        // deal with header line:
        getline( inFile, aString ) ;
        tokenize( aString, tokens ) ;
        assert( tokens[1]=="ID" ) ;
        assert( tokens[2]=="DEPTLINK" ) ;
        assert( tokens[3]=="ARRLINK" ) ;
        assert( tokens[4]=="TIME" ) ;
    }
    // always check if file is still open:
    assert( inFile.is_open() ) ;
```

```
    // main part:
    while ( !inFile.eof() ) {
        getline( inFile, aString ) ;
        if ( !aString.empty() && isdigit( aString[0] ) )
            // [[ skip lines with junk ]]
        {
            tokenize( aString, tokens ) ;
            Id travId ; convert( tokens[1], travId ) ;
            Id startLinkId ; convert( tokens[2], startLinkId ) ;
            Id endLinkId ; convert( tokens[3], endLinkId ) ;
            Time startTime ; convert( tokens[4], startTime ) ;
            set_travId( travId ) ;
            set_startLinkId( startLinkId ) ;
            set_endLinkId( endLinkId ) ;
            set_startTime( startTime ) ;
            set_nNodes( 0 ) ; // set number of node tokens to zero
            return 0 ;
        }
    }
    return 1 ; // return 1 when eof is encountered
}
```

Note that the methods to set the plans variables were already defined in Sec. 9.5.

Task 11.1 *Write a program that constructs the network, reads trips, and outputs them to the screen. Trips are at*

`http://www.matsim.org/files/studies/corridor/teach/0.trips .`

11.7 FindPath and Dijkstra

Remember that before calling the Dijkstra algorithm, the starting/ending locations which are on links need to be pushed forward/backward to the corresponding nodes. For us, links are always uni-directional, so that the answer to this is unique. This can look as follows:

```
int RouteWorld::FindPath ( Plan& plan ) {
    Link* startLink = links[plan.startLinkId()] ;
    assert( startLink != NULL ) ;
    assert( startLink->id()==plan.startLinkId() ) ;
    Link* endLink = links[plan.endLinkId()] ;
    assert( endLink!= NULL ) ;
    assert( endLink->id()==plan.endLinkId() ) ;
    Node* startNode = startLink->toNode() ;
    Node* endNode = endLink->fromNode() ;
    Dijkstra( startNode, endNode, plan.startTime() ) ;
    Node* tmpNode = endNode ;
    int cnt=0 ;
    while ( tmpNode != NULL ) {
        cnt++ ;
        tmpNode = tmpNode->prev() ;
    }
    plan.set_nNodes( cnt ) ;
    tmpNode = endNode ;
    for ( int ii=plan.lastNodeIndex(); ii>=plan.firstNodeIndex() ; ii-- ) {
        plan.set_nodeTokens( ii, tmpNode->id() ) ;
        tmpNode = tmpNode->prev() ;
    }
    return 0 ;
}
```

Note that this calls `Dijkstra`. The code after the `Dijkstra` call takes the Dijkstra algorithm result and copies it into `Plan`. `Plan.SetNNodes` sets the number of nodes the route traverses from the start link to the destination link. `Plan.SetNodeTokens` sets the corresponding tokens to the node IDs. An implementation for this was already given earlier (Sec. 9.5).

`Dijkstra` itself can look as follows. The precise meaning of `nodeList` will be described afterwards; essentially, it is a container that contains all "pending" nodes. In Sec. 11.2

this corresponds to the set of all nodes where `isDone` is false but `arrTime` is no longer infinity.

```
int RouteWorld::Dijkstra ( Node* startNode, Node* endNode, Time startTime ) {
    NodeList pending ;
    // general initialization:
    for ( Nodes::iterator nn=nodes.begin(); nn!=nodes.end(); nn++ ) {
        Node* theNode=nn->second ;
        theNode->unset_isDone() ;
        theNode->set_arrTime( INFTY ) ;
        theNode->set_prev( NULL ) ;
    }
    // initialize start node:
    startNode->set_arrTime( startTime ) ;
    pending.insert( make_pair( startTime, startNode ) ) ;
    // Dijkstra loop proper:
    while( pending.size() > 0 ) {
        Node* theNode = pending.begin()->second ;
        pending.erase( pending.begin() ) ;
        if ( !(theNode->isDone()) ) {
            // (check this because we may have nodes more than once in list)
            theNode->set_isDone() ;
            if ( theNode!=endNode ) {
                theNode->expand( pending ) ;
            } else {
                return 0 ;
            }
        }
    }
    // should never get here:
    assert(0==1) ;
}
```

The implementation for `NodeList` is again a multimap; the functioning of this was already explained in the context of generating a random sequence of links, and in the context of the vehicle wait queue. For the wait queue, the functionality is exactly the same has here: We need to maintain a set of (key,pointer)-pairs such that it is possible to retrieve the pointer which belongs to (one of) the smallest key(s).

One issue here is that, if a better `ArrTime` for a node is found, it should be moved within the priority queue. This would necessitate to find that element within the queue. Another option is to leave *both* entries in the queue, but add the `IsDone` flag to nodes. If a node with `IsDone` is encountered, it is removed from the queue but ignored otherwise.

The `expand()` method is still missing. Here is a suggestion:

```
void Node::expand ( RouteWorld::NodeList& pending ) {
    Time now = arrTime_ ;
    for ( VLinks::iterator ll=outLinks().begin(); ll!=outLinks().end(); ll++ ) {
                Link* link = *ll ;
        Node* nextNode = link->toNode() ;
        Time linkTTime = link->tTime( now ) ;
        Time nextTime = now + linkTTime ;
        if ( nextTime < nextNode->arrTime() ) {
            nextNode->set_arrTime( nextTime ) ;
            assert( !(nextNode->isDone()) ) ;
            nextNode->set_prev( this ) ;
            pending.insert( make_pair( nextTime, nextNode ) ) ;
        }
    }
}
```

`tTime( .  )` is a method of the `Link` class which returns the link travel time on that link as a function of the entering time, in the code given by `now`. As discussed in Sec. 11.3, at this point this should return the length of the link (in meters) divided by 7.5.

Task 11.2 *Run FindPath on the first activity in*

`http://www.matsim.org/files/studies/corridor/teach/0.trips`

Which route is returned? Why?

11.8 Plans output

Now the plan needs to be written to file. Since we have it already in a suitable internal representation, that is easy now:

```
int Plan::writePlan ( ) {
    static ofstream outFile;
    // open file if this is the first call:
    static int first=1 ; if ( first ) {
        first = 0 ;
        outFile.open(PLANS_FILE_NAME);
    }
    // always check if file is really open:
    assert( outFile.is_open() ) ;
    // fixed length part
    for ( int ii=1; ii<=18; ii++ ) {
        outFile << fixTokens_[ii] ;
        if ( ii==6 || ii==11 || ii==14 || ii==17 || ii==18 ) {
            outFile << endl ;
        } else {
            outFile << ' ' ;
        }
    }
    // variable length part
    for ( int ii=1; ii<=fixTokens_[18]; ii++ ) {
        outFile << varTokens_[ii] << ' ' ;
    }
    // Add an empty line:
    outFile << endl << endl ;
    return 0 ;
}
```

Task 11.3 *Apply your router to*

`http://www.matsim.org/files/studies/corridor/teach/0.trips`

and generate the corresponding plans file in Transims format. Note that the result is not *similar to*

`http://www.matsim.org/files/studies/corridor/teach/0.plans` .

Chapter 12

Congestion-dependent router

12.1 Link travel times and congestion

So far, the router is not sensitive to congestion. In order to make the routes sensitive to congestion, delays caused by congestion need to show up in the link travel times. This can be achieved via getting the link travel times from a separate file. Links which are congested will have link travel times which are longer than the free speed travel times.

In practice, we will achieve this via the events file. The events file, as discussed in Sec. 9.11, contains for each vehicle the time when it enters and the time when it leaves each link. We will aggregate this information as a function of the link entry times. The procedure consists of the following steps:

- **Conversion of events to link travel times.** For each enter-link-event, the corresponding leave-link-event is searched. As a result, one obtains for each link entry time a corresponding link travel time.

- **Aggregation.** Link travel times are aggregated into time slices, of e.g. 15 min. For this, the link travel times of all vehicles entering a link during a certain time slice are averaged. For example, if there are vehicles entering at 9:03:22, 9:05:56, and 9:07:23, and their link travel times are 1 min, 2 min, and 3 min, then the average link travel time for all vehicles entering between 9 and 9:14:59 will be 2 min.

  This type of data aggregation is the simplest method possible and it has certain drawbacks. This will be discussed in more detail in Sec. 19.1. **[[check if done]]**

Let us consider why this method works. The Dijkstra algorithm, as explained in Sec. 11.2, proceeds by "expanding" a node when no faster path to that node can be found. For that reason, the "current time" at that node, denoted by `now`, is the time-of-day when the node is reached via the fastest path. It is therefore also the time-of-day then the outgoing links from that node are entered.

Note: With time-dependence as explained above, it could happen that "waiting at a node" yields a faster path. This can happen when the link travel time in the following time bin is shorter than the link travel time in the current time bin plus the remaining time in the current time bin. In such a situation, the above algorithm would not return the path that is technically the fastest. In real traffic, however, this is rarely an issue: Links are approximately FIFO (first-in first-out), which means that entering at a later time also means leaving at a later time. In other words: If the time-dependent algorithm "thinks" that waiting at a node would pay off, then this is normally an artifact of the routing algorithm – more specifically, of the time aggregation – and not a feature of the traffic system. For those reasons, using the algorithm as described above will normally describe plausible routes, even if they may not be the technically fastest.

Yet, there is at least one situation where indeed waiting at a node could pay off: This is if links are opened at a certain time-of-day. We will not assume such complications here.

Implementation

12.2 Congestion dependency: Link travel times

We need to get the congestion information into the router. More specifically, we need that the correct link travel time information is returned by `link->tTime(now)` in Sec. 11.7.

As said above, the way we do this is by reading the events file, calculating each vehicle's link travel times, and then aggregating those times into the desired time bins. Here is a suggestion of a method to do this; comments are added below.

```
class EnterEvent {
private:
    Time time_ ;
public:
    void set_time( Time val ) { time_ = val ; }
    Time time() { return time_ ; }
private:
    Id linkId_ ;
public:
    void set_linkId( Id val ) { linkId_ = val ; }
    Id linkId() { return linkId_ ; }
private:
    Id vehId_ ;
public:
    void set_vehId( Id val ) { vehId_ = val ; }
    Id vehId() { return vehId_ ; }
} ;
void RouteWorld::readEvents () {
    cout << "\n### entering readEvents ..." << endl ;
    int cnt=0 ;
    // preprocessing (initialize Sum and Cnt):
    for ( Links::iterator ll=links.begin(); ll!=links.end() ; ++ll ) {
        Link* link=ll->second ;
        link->tTimeIni() ;
    }
    // open file:
    ifstream inFile ; inFile.open(EVENTS_FILE_NAME) ;
    assert( inFile.is_open() ) ;
    string aString ;
    vector<string> tokens ;
    // process header line:
    getline( inFile, aString ) ; tokenize( aString, tokens ) ;
    const int t_idx=1 ; assert( tokens[t_idx]=="TIMESTEP" ) ;
    const int v_idx=2 ; assert( tokens[v_idx]=="VEHICLEID" ) ;
    const int l_idx=3 ; assert( tokens[l_idx]=="LINK" ) ;
    const int n_idx=4 ; assert( tokens[n_idx]=="FROMNODE" ) ;
    const int f_idx=5 ; assert( tokens[f_idx]=="FLAG" ) ;
    typedef map<Id,EnterEvent*> EnterEvents ; EnterEvents enterEvents ;
    // main loop:
    while ( !inFile.eof() ) {
        getline( inFile, aString ) ;
        if ( !aString.empty() && isdigit( aString[0] ) ) {
            // ( skip lines w/ junk (e.g. last line) )
            tokenize( aString, tokens ) ;
            Time time ; convert( tokens[t_idx], time ) ;
            Id vehId ; convert( tokens[v_idx], vehId ) ;
            Id linkId ; convert( tokens[l_idx], linkId ) ;
            Id fromNodeId ; convert( tokens[n_idx], fromNodeId ) ;
            int flag ; convert( tokens[f_idx], flag ) ;
            if ( flag==ENTER_LINK_FLAG ) {
                EnterEvent* enterEvent = new EnterEvent ;
                enterEvent->set_time( time ) ;
                enterEvent->set_linkId( linkId ) ;
```

```
                enterEvent->set_vehId( vehId ) ;
                assert( enterEvents.count( vehId ) == 0 ) ;
                enterEvents[vehId] = enterEvent ;
            } else if ( flag==LEAVE_LINK_FLAG ) {
                EnterEvent* enterEvent = enterEvents[vehId] ;
                assert( enterEvent != NULL ) ;
                assert( enterEvent->linkId() == linkId ) ;
                Link* link = links[ linkId ] ;
                Time ttime = time - enterEvent->time() ;
                link->addToSum( enterEvent->time(), ttime ) ;
                cnt++ ;
                enterEvents.erase(vehId) ;
                delete enterEvent ;
            }
        }
    }
    if ( enterEvents.size() != 0 ) {
        cout << " severe warning: events map not empty " << endl ;
    }
    cout << " nEvents: " << cnt << endl ;
    cout << "### leaving readEvents ..." << endl << endl ;
}
```

Comments:

- In the initialization, all sums and count variables are set to zero via

  ```
  void Link::tTimeIni () {
      sum_.assign(maxBin_ + 1, 0);
      cnt_.assign(maxBin_ + 1, 0);
  }
  ```

  `sum_` and `cnt_` are vectors (e.g. `vector<int> sum` etc.). The `assign(N,X)` command sets elements `0` to `N-1` of the vector to value `X`.

  After that, the file is opened and the header line is read.

- In the main loop, the method goes through each line of the file, puts it into `aString`, checks for garbage, reads the corresponding values for time, vehicle id, link id, from-node id, and the event flag. If the event flag denotes an enter-link-event, then this information is added to a map with the vehicle id as key. ***Note that for this the vehicle id needs to be unique.*** If the event flag denots a leave-link-event, then the corresponding enter-link-event is retreived, the link travel time is computed, and it is added to the relevant time bin. The latter is achieved by

  ```
  void Link::addToSum ( Time now, double sum ) {
      unsigned bin = timeToBin( now ) ;
      assert( bin < sum_.size() ) ;
      sum_[bin] += sum ; cnt_[bin] ++ ;
  }
  ```

  This uses

  ```
  int timeToBin ( Time theTime ) {
      return int( theTime/900 ) ;
  }
  ```

The correct link travel time is now returned by

```
Time Link::tTime ( Time now ) {
    unsigned bin = timeToBin( now ) ;
    assert( bin < sum_.size() ) ;
    if ( cnt_[bin] > 0 ) {
        return Time( sum_[bin]/cnt_[bin] ) ;
    } else {
        return Time( length()/GBL_FREE_SPEED ) ;
    }
}
```

Note that this uses the free speed travel time if no events information is available. Here, we use the global variable `GBL_FREE_SPEED`; this could be replaced by link-dependent free speeds in more sophisticated implementations. However, when doing this, one needs to make sure that also the traffic simulation generates link-dependent free speeds. Our simulation of Chap. 7 does not do this; improving this will be discussed in Chap. 17.

It is useful to note that all conversions from time-of-day to time-bins is done via the function `timeToBin`. The inverse conversion (from time bins to time-of-day) is never needed. This makes sure that if the router requests information for a certain time-of-day, it will *always* receive the same time bin that a link entry event at the same time would have obtained.[1]

Clearly, the overall integration into the router has to look as follows:

```
int main() {
    // instantiate routeWorld:
    RouteWorld routeWorld ;

    // read the network:
    routeWorld.readNodes() ;
    routeWorld.readLinks() ;

    // read the events:
    routeWorld.readEvents() ;

    // main loop:
    ...
}
```

Task 12.1 *Write routines which read the events. Check if the processing of*

`http://www.matsim.org/files/studies/corridor/teach/test.events`

leads the link travel times would expect. (Which values would you expect?)

Task 12.2 *Run FindPath together with*

`http://www.matsim.org/files/studies/corridor/teach/test.events`

on the first trip in

`http://www.matsim.org/files/studies/corridor/teach/0.trips`

Which route is returned? Is this different from the route returned in Task 11.2? Why?

Task 12.3 *Get the events file that was produced by running the traffic micro-simulation on*

`http://www.matsim.org/files/studies/corridor/teach/0.plans`

Read those events, and then apply your router to

`http://www.matsim.org/files/studies/corridor/teach/0.trips`

Give the resulting routes file to the micro-simulation and have it executed. Does the result make sense? Why or why not?

[1] Earlier versions, by Transims and also by ourselves, aggregated the event information into the time bins either directly in the traffic simulation, or by some external module, and wrote the result into a file. The typical information given in that file was a time, say "900 sec", and a corresponding link travel time. In implementations, there was then always confusion if this referred to a time bin going from 1 to 900, or to a time bin going from 900 to 1799. The intention was the first, but unfortunately `time%900` (where `%` is the modulo function) puts 0 to 899 into one time bin and 900 to 1799 into another one, resulting in many errors. Clearly, this is a trivial problem, but one that continuously caused problems.

Chapter 13

Feedback/System integration

13.1 Introduction

As explained in Chap. 2, "learning" or "adaptation" is an extremely important part of transportation simulations packages. The idea is that if the execution of a plan differs from what people had expected, then they will change their plans to adapt to what they found. For example, if congestion lets them arrive late to work, they will leave home earlier.

We will implement this in a very straightforward way: The traffic simulation will collect link travel times, and the router will use them to generate better routes. This reflects **day-to-day learning**, that is, travelers revise their decisions from one day to the next. This is in contrast to **within-day learning**, which will be treated later.

We will also allow only 10% of the travelers to replan between any given two days, in order to avoid over-reactions of the system. Such over-reactions could otherwise for example happen if alternative A was slightly faster than another one in one iteration and as a result *all* travelers would switch to link A, making it extremely congested. There are other ways to deal with this problem, which will also be treated later in the class.

Fig. 13.1 gives information about the data flow through the different elements.

Implementation

13.2 Subset of trips file

You want the router to compute new routes only for 10% of the travelers. For this, you need to generate a random sample of the trips file. Do the following:

- Write the trips file header.
- For each traveler in the trips file, decide if that traveler should be re-planned. If yes, write the trip line into the new file.

Awk is a good language for parsing line-oriented files, which is why we introduce it here.

```
BEGIN {
    # print header line of trips file
    print "ID" , "DEPTLINK" , "ARRLINK" , "TIME", "NOTES" ;
}
{
    # Skip header line and comments:
    if ( $1 == "#" || $1 == "ID" ) { next; }
```

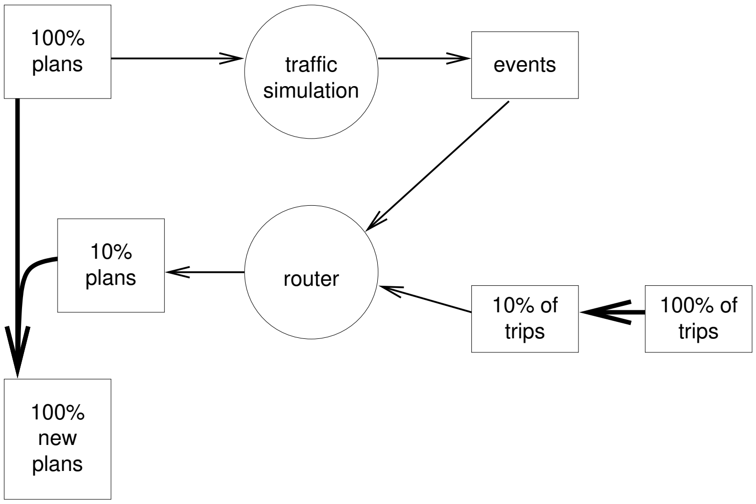


Figure 13.1: Data flow through the simple feedback mechanism of this chapter. Reading the network files is not drawn. The thick lines are the ones which need to be done in this Chapter.

```
      # w/ proba 10%, write out the line again:
      if ( rand() < 0.1 ) {
          print $0 ;
      }

  }
```

If the above is called `SelectTrips.awk`, then is is called via

```
gawk -f SelectTrips.awk < 0.trips > 1.trips
```

The code consists of three parts:

1. An optional “BEGIN” block. This is executed before anything is read.
2. A block without special identifier. For every line out of `test.events`, this block is executed.
3. An optional “END” block. This is executed just before the program is exited.

See “man awk” for more information.

IMPORTANT: Make sure you use different random seeds every time you call this module, otherwise the same 10% travelers get replanned over and over again.

In awk, “rand()” returns a random number. See “man awk”.

Task 13.1 *Generate a set of 10% randomly selected trips. Use*

```
http://www.matsim.org/files/corridor/teach/0.trips
```

as input.

13.3 Calling the router

You should now be able to call the router. Make sure that the router really reads the files (events, trips) that you provide. For this, it is recommended to re-do task 11.2 and check if the router truly responds to the files you give to it.

Task 13.2 *Generate a set of routes which have responded to congestion.*

13.4 Merging of the routes

Now you have two files with routes, one with the old routes for all travelers, and one with the new routes for 10% of the travelers. We need to merge them.[1] For the merging, we can assume that the plans are in order, since they are generated from the same trips file. So you have to write code which does the following:

- Open both files, old.plans and new.plans.
- Read the first plan from each file.
- If they have the same traveler id, then
  - discard the old plan and write the new plan into merged.plans.
  - Read the next plan from each file, and continue.
- If they do not have the same traveler id, then
  - write the old plan into merged.plans.
  - Read the next plan from old.plans, and continue.

Note that you could use ReadPlans and WritePlans from Secs 9.4 and 11.8. Awk does not work so well here since the format is not line oriented.

13.5 Traffic simulation

Task 13.3 *Now you should run the traffic simulation on the new plans set. Make sure (e.g. in Vis) that some travelers really use new routes (0.plans has all traffic on the middle road). This is called the 1st iteration. When does the last vehicle leave your simulation?*

13.6 Iterations

Now we want to do systematic iterations. You should write a script which manages those iterations. One option is perl; shell scripts work well, too. Also, some clever Makefile writing is an option. The script does the following:

- Run the usim on a given plans file.
- Generate a random 10% trips file.
- Run the router on the 10% trips file using the events from the last simulation.
- Merge the plans.
- Run the usim again.
- Etc.

Task 13.4 *Do 50 iterations. Keep all information (routes, events, snapshot files) for every 10th iteration.*

Keep events files for all *iterations.*

Compress (e.g. gzip) all output files.

Task 13.5 *Plot the sum of all vehicle travel times as a function of the iteration number.*

Note that you can derive this information from the events files.

[1] This is truly awkward. In our research, we put the new plans into a data base, which keeps track of *all* plans. Then we dump out the plans we want. That solution is much cleaner, but besides being more difficult to implement, it is also slow, so it is not the final answer.

Chapter 14

Activities planner: Adjust trip starting times

14.1 Introduction

So far, we have a traffic micro-simulation module, and a routing module. The input to all this, apart from the network information, are the trips. However, these trips need to be generated somehow. As a first step towards this, we will consider the question of departure time choice. Let us assume that people want to arrive at work at a particular time. There is a penalty associated with being early (which consists of wasted time), and a penalty associated with being late (which may consist of an angry employer). Also, the travel time may vary depending on when one travels. The idea is that there is a trade-off between these elements. For example, if the travel time is much shorter when traveling early, people may accept being early in spite of the waste of time. This is in particular true if one has a time window to start work, and the only argument against starting early is that one has to get up early.

14.2 Utilities

14.2.1 Basic idea

These trade-offs are operationalized via giving utitilies to the different aspects of the situation. The utilities in this chapter will be negative, which is why they are sometimes called disutilities. Let us assume that we have the following utilities:

- The (dis)utility of the trip time, $U_{trip}(T_{trip})$. It depends on the trip time, T_{trip}.
- The (dis)utility of being early, $U_{early}(T_{early})$. It depends on how early the traveler is. If the traveler is late, this contribution is zero.
- The (dis)utility of being late, $U_{late}(T_{late})$. It depends on how late the traveler is. If the traveler is early, this contribution is zero.

Let us further assume that these utilities are additive (see Fig. 14.1):

$$U_{dep} = U_{trip}(T_{trip}) + U_{early}(T_{early}) + U_{late}(T_{late}) \; . \tag{14.1}$$

An example is:

$$U_{dep} = -\frac{0.4}{60\;sec} T_{trip} - \frac{0.25}{60\;sec} T_{early} - \frac{1.5}{60\;sec} T_{late} \; . \tag{14.2}$$

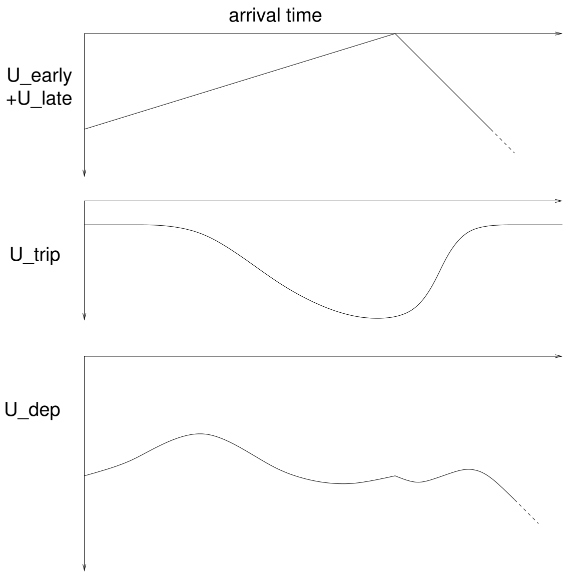


Figure 14.1: Utility contributions

The results of this come out in arbitrary utility units, sometimes called “utils”.

14.2.2 Dependence on departure time

Fig. 14.1 gives the function of the different utilities as a function of the *arrival* time. For the calculation that we will do later, we need them as a function of *departure* time. For example, if t_{des} is the desired arrival time, then

$$T_{early}(t_{dep}) = \max\Big(0, t_{des} - t_{early}\Big) = \max\Big(0, t_{des} - (t_{dep} + T_{trip})\Big) . \tag{14.3}$$

Here, T_{trip} again depends on t_{dep}, and therefore

$$T_{early}(t_{dep}) = \max\Big(0, t_{des} - (t_{dep} + T_{trip}(t_{dep}))\Big) . \tag{14.4}$$

As we will see later, we will essentially need a *table* of the values of T_{early} as a function of t_{dep} where t_{dep} increases in 5-min time steps. Because of this simplification, the problem can be solved as a sequence of look-ups, resulting in a table similar to the following (where $t_{des} = 8:00$)

t_{dep}	$T_{trip}(t_{dep})$	$T_{early}(t_{dep})$
6:00	0:15	1:45
$\vdots$		
7:00	0:15	0:45
7:05	0:19	0:36
7:10	0:30	0:20
$\vdots$		

14.3 Departure time selection

In general, one would assume that travelers select the departure time with the largest utility. Let us however assume that the above utility calculation is somewhat fuzzy, for example because travelers do not know the different contributions exactly. Then, we want that the probability to select a certain departure time grows with the respective utility.

A typical mathematical form to achieve this if one has to select between several different options i is

$$p_i \propto e^{\beta U_i} \ . \tag{14.5}$$

Since p_i is a probability, this needs to be normalized, i.e. one wants $\sum_i p_i = 1$, where the sum goes over all possible options. This results in

$$p_i = \frac{e^{\beta U_i}}{\sum_j e^{\beta U_j}} \ , \tag{14.6}$$

where the sum in the denominator goes over all possible options including i.

Note that this mathematical form does exactly what we want: if U_i is large, then option i has a high probability of being selected. The parameter β changes the randomness of this choice.

- If $\beta \to 0$, then the choice does not depend on the U_j; in consequence, it is totally random with equal weight on each option.
- If in contrast $\beta \to \infty$, then the option with the highest utility will be selected with probability one, and all others will never be selected.

  One way to see this is the following. Assume that U_{max} is the largest utility, and let us assume that there is only one optimal choice (to simplify the argument). First let us look at a non-optimal choice i, i.e. $U_i < U_{max}$. Then

  $$p_i = \frac{e^{\beta U_i}}{e^{\beta U_{max}} \sum_j e^{\beta (U_j - U_{max})}} < \frac{e^{\beta U_i}}{e^{\beta U_{max}}} \ , \tag{14.7}$$

  since the sum is larger than one. (One of the contributions comes from $U_j = U_{max}$, and all other contributions are positive.) This can be rewritten as

  $$e^{\beta (U_i - U_{max})} \stackrel{\beta \to \infty}{\longrightarrow} 0 \tag{14.8}$$

  (because $U_i - U_{max} < 0$).

  Now let us look at the optimal choice k, i.e. $U_k = U_{max}$. Then

  $$p_k = \frac{1}{\sum\limits_j e^{\beta (U_j - U_{max})}} = \frac{1}{e^{\beta \cdot 0} + \sum\limits_{j \neq k} e^{\beta (U_j - U_{max})}} \stackrel{\beta \to \infty}{\longrightarrow} \frac{1}{1+0} \ , \tag{14.9}$$

  because $U_j - U_{max} < 0$ for $j \neq k$.

14.4 Operationalization

Departure time choice will be operationalized in the following way. We will take Eq. (14.2) as an example, and set $\beta = 1$. Let us in addition decide that we look at 5min time bins, and that we consider times only between 5am and 10am. Let us consider a traveler who wants to arrive at t_{des}.

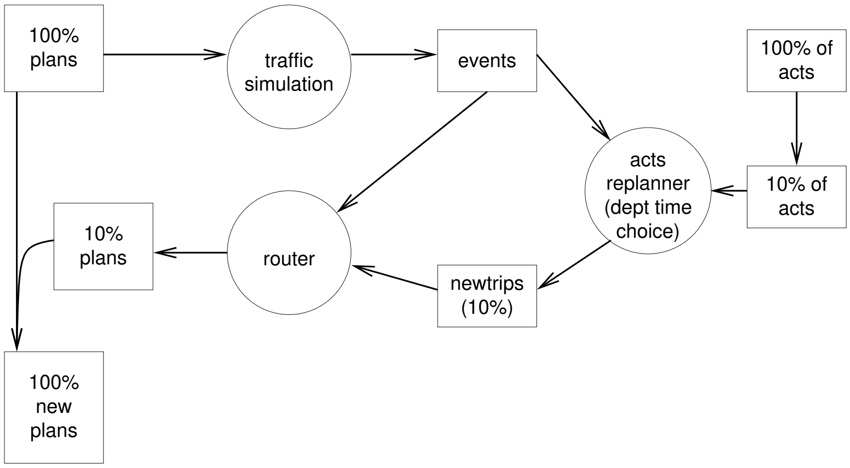


Figure 14.2: Data flow for simple activities replanning.

This traveler would calculate, for all times between 5am and 10am in 5min time steps, and for her/his desired arrival time t_{des}, the value $f(t_{dep}) = e^{U(t_{dep})}$. She/he would then calculate the sum of all these values, Σ. The probabilities would then come out as

$$p(t_{dep}) = \frac{f(t_{dep})}{\Sigma} \; . \tag{14.10}$$

The traveler would then randomly select one of these departure time options according to the weights given by Eq. (14.10).

The data flow for activities replanning is given in Fig. 14.2. Note that travelers with new departure times also get new routes. At this point we do not perform separate re-routing for travelers whose activities have not changed. **[[This will be changed in Chap. ??.]]**

Implementation

14.5 Input data: Activities file

Demand for travel (= trips) is driven by activities taking place at different locations. We encapsulate this fact into a simple activities file, as follows:

Column	Header	type	explanation
1	**TRAV_ID**	**integer**	**ID number of traveller/vehicle**
2	**ACT_TYPE**	**string**	**type of the activity (“h” = home, “w” = work)**
3	**LINK**	**integer**	**activity location (link ID)**
4	**DES_ARR_TIME**	**integer**	**desired arrival time at activity**
5	NOTES	string	notes (optional)

An example is in

`http://www.matsim.org/files/studies/corridor/teach/0.acts .`

For our work here, we will assume that activities always come in pairs, i.e. that each individual in the simulation starts at one location (“at home”) and goes to another location (“work”). We also assume that there is a desired arrival time for the work activity.

Task 14.1 *Write a utility (e.g. using awk) that generates a new activity file which consists of a randomly selected 10% of the input activity file. This will be needed later.*

14.6 Origin-destination travel times

For the computation of departure time choice, one needs information about the trip times as a function of different departure times. **[[Different possibilities are discussed in** you assume that you have T time bins, R origins, and S destinations, then this results in $T \times R \times S$ entries.

Task 14.2 *Write a script that averages OD travel times into 15-min time bins. Language possibilities are awk or c++/java. As an end result, you should have, for all OD pairs, trip time info for all 15-min time bins. Generate this information for the events file which was obtained by running the traffic microsimulation on*

`http://www.matsim.org/files/studies/corridor/teach/0.plans .`

Why does the result make sense (or not)?

Note that you have to invent some method to generate OD travel times for time bins for which you have no information.

14.7 Departure time choice

Now the departure time needs to be chosen for each individual traveler. For this, it is easiest to continue with the code written in Sec. 14.6 (Task 14.2). After retrieving the travel time information from the events file, the code will start reading the 10% activities file produced in Sec. 14.5. For each agent it will retreive a *pair* of activities. The desired arrival time t_{des} comes from there as discussed above. For each activity pair in the activities file do:

1. Retrieve or calculate, for each departure time t_{dep} between 5am and 10am in 5min steps, the following quantities:
   - the trip time T_{trip};
   - the arrival time t_{arr};
   - the early time $T_{early} = \max[0, t_{des} - t_{arr}]$;
   - the late time $T_{late} = \max[0, t_{arr} - t_{des}]$;
   - the resulting utility
   $$U_{dep} = -\frac{0.4}{60\ sec} T_{trip} - \frac{0.25}{60\ sec} T_{early} - \frac{1.5}{60\ sec} T_{late} \tag{14.11}$$
   (this is the same as Eq. (14.2));
   - and the resulting non-normalized probability
   $$\pi_i = e^{U_{dep}} \ . \tag{14.12}$$

2. Once you have done this for all time bins, sum up all the non-normalized probabilities:
   $$\Pi := \sum_i \pi_i \ . \tag{14.13}$$
   Divide all non-normalized probabilities by this value:
   $$p_i := \pi_i / \Pi \ . \tag{14.14}$$

3. Make a random draw between these probabilities (see below) and note the resulting departure time.
4. Fuzzify the departure time by ± 150sec (2.5min) by something like
   ```
   TDepInSec = TDepInSec - 150 + int( 300*MyRand() ) ;
   ```
5. Write out the corresponding trip.

All trips then need to be routed; this is done by applying the time-dependent router to the trips file as before.

We need to make a random draw according to the probability weights. This is for example done as follows. Assume that we have `p[i], i=1..N` given, with the sum of these $p[i]$ being one. Then do something like the following:

```
double rnd = myRand() ;
double sum = 0. ;
int ii ;
for ( ii=1; ii<=N; ii++ ) {
     sum += p[ii] ;
     if ( sum > rnd ) break ;
}
// ii is the desired index.
```

Task 14.3 *Take the events file from the 50th iteration of the corridor problem. Generate, for travelers 1–250 in*

`http://www.matsim.org/files/studies/corridor/teach/0.acts ,`

the departure times (= new trips). Plot the resulting new departure time distribution (see below). Does this correspond to your expectations? Why (or why not)?

Note: Departure time distribution means that on the x-axis you have the departure time, and on the y-axis you have how many vehicles/travelers depart at that time. For this, you again need to introduce time bins, for example 5 minutes wide.

14.8 Feedback

Task 14.4 *Do 100 iterations. Make the following plots:*

- *Sum of all trip times as function of iteration number.*
- *Computing time*
- *veh.bin files for days 1, 10, 20, 100.*
- *Departure time distribution for days 1, 10, 20, 100.*[1]

Is the final departure time distribution plausible? Why (or why not)?

Task 14.5 *Question: Is is possible that everybody finds a departure time so that she/he arrives exactly at her/his desired arrival time?*

[1] We are looking for the departure time distribution of the *whole* population, not just of the replanned population. This is best retrieved from the events file.

Chapter 15

Do-it-yourself transportation planning simulation: Summary

The previous chapters have led you through a do-it-yourself version of a transportation planning simulation. Irrespective of the fact if you have really implemented all of it, or just pieces, or none at all, several things should have become clear:

- Transportation simulations do not only consist of the traffic modules, where cars and people move through the system, but also of strategic/tactical modules which simulate the human decision-making that generate the traffic in the first place.
- Although a whole transportation simulation package is a complex software system, programming a "lite" version that concentrates on the most important aspects is a manageable task.
- Modern computer science tools, in particular object-oriented programming languages, are very helpful for programming these types of simulations. The challenge is to find a good balance between where these additional language features really help and where they make things uncomprehensible to the uninitiated.

These past chapters have attempted to concentrate on the bare-boned essentials. Clearly, what is essential and what not depends on one's preferences and taste. The focus of this text is on the *multi-agent* view, i.e. the fact that a transportation simulation can be seen as a simulation of many intelligent, interacting agents. In consequence, we have stressed that all individual travelers make their individual plans, and that these plans can be revised in iterated simulations – in other words, the agents learn. The underlying traffic simulation, a 1-lane cellular automata simulation, was designed such that it could execute individual plans in a meaningful way, but it was not attempted to make that simulation realistic.

The following chapters of this text will show how that simulation can be improved. Improvements are primarily into two directions: (i) more realism; (ii) truly agent-based view. These aspects will be discussed in more detail in the introduction to Sec. ??.

[[following goes where??]]

- More realism. In particular the traffic simulation can be made much more realistic. We will first show one version (the queue simulation) which is both more realistic and computationally much faster; it however models traffic on a higher level of abstraction which is sometimes more difficult to grasp. Higher levels of realism are also introduced for the router (time dependence, other modes of transportation), and, to some extent, for activity generation. All these are researched intensely, since multi-agent simulation has opened the way to new exciting possibilities.

- Truly agent-based view. The simulation described in the last chapters depends on file-based interfaces, and these interfaces imply that the sequencing of the simulation is organized around modules. In general, modules will run sequentially, each module modifying some aspect of the system state that is displayed by the collection of input and output files. One will however easily recognize that this organization of the simulation is not truly agent-based, that is, the agent is not truly at the center. For example, programming an agent that uses mutation and crossover to create new strategies from the ones it has already tried out is awkward with the described framework.

Chapter 16

File formats summary

16.1 Nodes file

Column	Header	type	explanation
1	**ID**	**integer**	**Unique number of node**
2	**EASTING**	**integer**	**Coordinate in x direction**
3	**NORTHING**	**integer**	**Coordinate in y direction**
4	ELEVATION	integer	Coordinate in z direction. Ignore
5	NOTES	string	Optional notes. Ignore

16.2 Links file

Column	Header	Type	Explanation
1	**ID**	**integer**	**Unique ID number**
2	NAME	string	Name of the link, e.g. the street name. Ignore
3	**NODEA**	**integer**	**Node ID at one end of link**
4	NODEB	**integer**	**Node ID at other end of link**
5	PERMLANESA	integer	Number of lanes towards A. Ignore
6	PERMLANESB	integer	Number of lanes towards B. Ignore
7	LEFTPCKTSA	integer	Number of left pocket lanes towards A. Ignore
8	LEFTPCKTSB	integer	Number of left pocket lanes towards B. Ignore
9	RGHTPCKTSA	integer	Number of right pocket lanes towards A. Ignore
10	RGHTPCKTSB	integer	Number of right pocket lanes towards B. Ignore
11	TWOWAYTURN	boolean	Whether there is a two-way link for left turns in the middle of the road (an American specialty). Ignore
12	**LENGTH**	**positive float**	**Length of link in meters**
13	GRADE	float	Grade (= slope) of link. Ignore
14	SETBACKA	positive float	Setback distance (in meters) from the center of the intersection at node A. Ignore

15	SETBACKB	positive float	Setback distance (in meters) from the center of the intersection at node B. Ignore
16	CAPACITYA	positive float	Capacity of link towards A in vehicles per hour. Ignore (but see Sec. 18)
17	CAPACITYB	positive float	Capacity of link towards B in vehicles per hour. Ignore (but see Sec. 18)
18	SPEEDLMTA	positive float	Speed limit, in meters per second, towards A. Ignore (but see Secs. 17 and 18)
19	SPEEDLMTB	positive float	Speed limit, in meters per second, towards B. Ignore (but see Secs. 17 and 18)
20	FREESPDA	positive float	Free speed, in meters per second, towards A. Ignore (but see Secs. 17 and 18)
21	FREESPDB	positive float	Free speed, in meters per second, towards B. Ignore (but see Secs. 17 and 18)
22	FUNCTCLASS	keyword	Functional class of link. Ignore
23	THRUA	integer	ID of outgoing link across A which denotes "through" direction. Can be used for data compression. Ignore
24	THRUB	integer	ID of outgoing link across B which denotes "through" direction. Can be used for data compression. Ignore
25	COLOR	integer	Obsolete. Ignore
26	VEHICLE	keywords	Allowed modes on link. Ignore
27	NOTES	string	Arbitrary notes. Ignore

16.3 Snapshot file (visualizer output)

Column	Header	type	explanation
1	**VEHICLE**	**integer**	**Vehicle ID**
2	**TIME**	**integer**	**Current time (in seconds past midnight)**
3	**LINK**	**integer**	**Link ID**
4	**NODE**	**integer**	**FromNode ID (i.e. ID of node where the vehicle is coming from)**
5	**LANE**	**integer**	**Lane the vehicle is on**
6	**DISTANCE**	**float**	**Distance (in meters) the vehicle is away from the node**
7	VELOCITY	float	Vehicle speed (in meters per second)
8	VEHTYPE	integer	Vehicle type. "1" = car.
9	ACCELER	float	Vehicle acceleration (in m/s per second)
10	DRIVER	integer	Driver ID
11	PASSENGERS	integer	Number of passengers in vehicle
12	**EASTING**	**float**	**Position of vehicle in x direction**
13	**NORTHING**	**float**	**Position of vehicle in y direction**
14	ELEVATION	float	Position of vehicle in z direction

15	**AZIMUTH**	**float**	**Vehicle's orientation (degrees from east in counterclockwise direction)**
16	USER	integer	User-defined data field

16.4 Plans file

Fixed length part:

Number	explanation
1	**Traveler (Person) ID**
2	User field. Irrelevant for us
3	Trip ID. Irrelevant for us
4	Leg ID. Irrelevant for us
5	FirstLegFlag. Irrelevant for us
6	LastLegFlag. Irrelevant for us
7	**StartTime**
8	**StartLocation. = StartLink for us**
9	Type of StartLocation. Irrelevant for us
10	EndLocation. Irrelevant for us
11	Type of EndLocation. Irrelevant for us
12	Duration. Irrelevant for us
13	Stop Time. Irrelevant for us
14	MaxTimeFlag. Irrelevant for us
15	Driver Flag. Irrelevant for us
16	Mode. Should always be 0
17	Vehicle Type. Irrelevant for us
18	**Number of additional tokens (variable length part)**

Variable length part:

number	explanation
1	Vehicle ID. Ignore
2	Number of Passengers. Needs to be zero (because the meaning of the following data depends on this).
3	**Node 1**
4	**Node 2**
5	**etc.**

16.5 Events file

Column	Header	type	explanation
1	**TIMESTEP**	**int**	**time step**
2	**VEHICLEID**	**int**	**vehicle id**
3	**LINK**	**int**	**Link ID**
4	FROMNODE	int	FromNode ID for link. Irrelevant for us since we use uni-directional links
5	**FLAG**	**int**	**0: vehicle arrives at final destination** **2: vehicle leaves a link to go across an intersection**

			4: vehicle moves from wait queue into traffic **5: vehicle enters a link coming from an intersection** **6: vehicle is supposed to start**
6	NOTES	string	notes (leave empty, but separate by tab)

16.6 Trips file

Column	Header	type	explanation
1	**ID**	**integer**	**ID number of traveller/vehicle**
2	DEPTLINK	**integer**	**departure location (link ID)**
3	**ARRLINK**	**integer**	**arrival location (link ID)**
4	**TIME**	**integer**	**departure time of traveller/vehicle in "seconds past midnight"**
5	NOTES	string	notes (leave empty, but separate by tab)

16.7 Activities file

Column	Header	type	explanation
1	**TRAV_ID**	**integer**	**ID number of traveller/vehicle**
2	**ACT_TYPE**	**string**	**type of the activity ("h" = home, "w" = work)**
3	**LINK**	**integer**	**activity location (link ID)**
4	**DES_ARR_TIME**	**integer**	**desired arrival time at activity**
5	NOTES	string	notes (optional)

Part III

Improvements

Chapter 17

More realistic CA traffic simulation logic

17.1 Introduction

The focus of this whole text is to emphasize the modular structure of transportation simulation packages, and in particular that besides the movement of the cars through the system considerable effort needs to be spent on modules which model human learning and decision-making, and on mechanisms which couple those modules. In consequence, we have started (in Chap. 7) with a simple micro-simulation which is able to support our approach, which means that it has individual vehicles which follow individual plans. However, the simple approach of Chap. 7 neither looks at correct vehicle speed not at correct link flow capacities.

In this chapter, it will be discussed how the CA traffic simulation from Chap. 7 can be made more realistic. In fact, this type of simulation is used in the Transims simulation package for transportation planning. Ultimately, also the CA approach has its limits and is better replaced by an approach where the spatial coordinates are continuous (Chap. ??). The CA approach has however the advantage that its implementation is rather straightforward. This is due to the simple spatial structure, in which the existence of a vehicle at a specific location can be checked via a simple direct lookup at the corresponding cell. Techniques with continuous coordinates typcally store the position of the particle together with the particle, i.e. *not* together with the spatial substrate, so that the existence of vehicles at specific locations needs to me made computationally efficient via other methods. These problems can be overcome, and the resulting models are as efficient as CA models, but they represent some conceptual and programming overhead that needs to be recognized.

17.2 The stochastic traffic cellular automaton (STCA)

The CA introduced in Chap. 7 can be made more general by allowing vehicles to travel more than one cell per time step. Also, it makes the simulation more realistic and more robust against artifacts if one introduces some randomness. Both are achieved with the following update rules:

- **Car-following rule:**

$$v_{safe} = \min\{v_t + 1, g_t, v_{max}\} \ . \tag{17.1}$$

g_t is the number of empty spaces to the car in front ("gap"); v_{max} is the maximum velocity of the car under consideration.

- **Randomization:**

$$v_{t+1} = \begin{cases} \max\{v_{safe} - 1, 0\} & \text{with probability } p_n \\ v_{safe} & \text{else} \end{cases} \tag{17.2}$$

- **Moving:**

$$x_{t+1} = x_t + v_{t+1} \tag{17.3}$$

t and $t + 1$ here refer to the actual time-steps of the simulation. The first rule describes deterministic car-following: try to accelerate by one velocity unit except when the gap is too small or when the maximum velocity is reached.

The second rule describes random noise: with probability p_n, a vehicle ends up being slower than calculated deterministically. This parameter simultaneously models three effects:

1. Speed fluctuations during free driving: Assume a vehicle with no other vehicles are nearby. It will eventually have speed $v_{max} - 1$ or v_{max}. In both cases, v_{safe} will be v_{max}. After the randomization, the speed will be at $v_{max} - 1$ with probability p_n, and at v_{max} else. That is, the speed of a single undisturbed vehicle fluctuates between v_{max} and $v_{max} - 1$.

2. Over-reactions at braking and car-following: Assume a vehicle with v_{max} that approaches a slower vehicle from behind. Eventually, it will reach a gap $g_t < v_{max} - 1$. v_{safe} will be equal to this g_t, and v_{t+1} will either be equal to g_t or one smaller (without becoming negative). That is, with probability p_n, the braking vehicle will not be at speed g_t but slower.

   The argument for car following is similar: Assume a leading vehicle with speed $v_{lead} < v_{max}$. The follower will attempt to follow with $g_t = v_{lead}$ but in fact will fluctuate around that speed.

3. Randomness during acceleration: Assume a single vehicle with speed zero. Instead of acceleration $0 \to 1 \to 2 \to 3 \to \ldots$, the acceleration will typically look like $0 \to 0 \to 1 \to 2 \to 2 \to 3 \to \ldots$. Note that the rules are such that the velocity never *de*creases during acceleration.

Obviously, these effects overlap to a certain extent; for example, if $g_t = v_{max}$ one cannot say if p_n refers to car following or to driving at free speed.

A translation into real-world units can be obtained as follows: The length ℓ of a cell is given by the average space a car occupies in a jam, since under jammed conditions each cell is filled by one car. Thus, $\ell = 1/\rho_{jam} \approx 7.5\ m$. A simulation time step typically corresponds to one second in reality, and the order of magnitude of this can be justified by reaction time arguments (Sec. 27.4.1). One of the side-effects of this convention is that space can be measured in "cells" and time in "time steps", and usually these units are assumed implicitly and thus left out of the equations. A speed of, say, $v = 5$, means that the vehicle travels five cells per time step, or 37.5 m/s, or 135 km/h, or approx. 85 mph.

p_n is often set to $1/2$ for theoretical work, while for realistic traffic modelling $p_n = 0.2$ is a better choice.

[[would be possible to show this in validation (more fdiags, as function of params)]]

17.3 Some validation of the STCA

Despite somewhat unrealistic features on the level of individual vehicles, these models describe aspects of the macroscopic behavior correctly. If we assume the values given above, i.e. a cell size of $\ell = 7.5\ m$ and a time step of $\Delta t = 1\ sec$, then speeds are given in multiples of $7.5\ m/sec = 27\ km/h = 16.875\ mph$. More correctly, average free speed is given by $(1 - p_{noise})\, v_{max}$. With $p_{noise} = 0.2$, one obtains the following possible average link speeds:

v_{max}	$v_{max} - p_{noise}$	m/sec	km/h	mph
1	0.8	6.0	21.6	13.500
2	1.8	13.5	48.6	30.375
3	2.8	21.0	75.6	47.250
4	3.8	28.5	102.6	64.125
5	4.8	36.0	129.6	81.000
6	5.8	43.5	156.6	97.875
7	6.8	51.0	183.6	114.750

Since drivers typically do not observe speed limits exactly, it is uncritical that these speeds do not correspond to any "round" numbers. Also, there is enough flexibility to model differences between, e.g., residential streets, urban arterials, freeways with speed limits, and freeways without speed limits. There is however not enough resolution to model, say, the difference between a speed limit of 60 vs. 65 mph. If such differences are of interest, a different model needs to be selected.

A typical measurement for real-world traffic is the flow-density fundamental diagram. For this, one measures flow and density at a fixed location over fixed periods of time, for example over 5 minutes. The resulting data is plotted with density on the x-axis and flow on the y-axis (see Fig. 17.1). There are some subtleties involved with measuring fundamental diagrams, which are discussed in Sec. 27.2. For the purposes of this section, let us assume that the two quantities are measured in the CA as follows:

- **Flow**: Count the number of vehicles, N_q, that cross a given location during time T. Flow q_T is given as

$$q_T = \frac{N_q}{T} \ . \tag{17.4}$$

- **Density:** Assume a "measurement area" which spreads across v_{max} contiguous cells. Sum up the number of vehicles on the measurement area over T time steps. This includes that a vehicle that spends more than one time step on the measurement area is counted several times. If this number is N_ρ, then density is given as

$$\rho_T = \frac{N_\rho}{T\, v_{max}} \ . \tag{17.5}$$

  Note that using v_{max} cells makes sure that every vehicle is counted at least once.

  The result is the density in "number of vehicles per cell", corresponding to "number of vehicles per 7.5 meters". Multiplying by $1000/7.5$ converts this into "number of vehicles per kilometer".

Flow-density fundamental diagrams, as in Fig. 17.1, start at zero flow when the density is zero (no cars on the road), and eventually come back to zero flow when the jam density is reached. In between, they show a roughly tri-angular shape as can be seen in Fig. 17.1. Theoretical discussions will be postponed until Chap. **??[[cha:traffic-flow-theory]]**, but it is important to note that there is some value of maximum flow, about $2000\ veh/h$ in

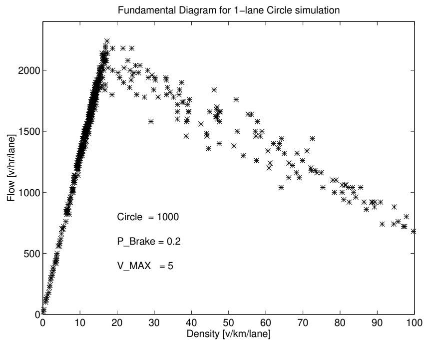


Figure 17.1: One-lane fundamental diagram as obtained with the standard cellular automata model for traffic using $p_{noise} = 0.2$. From (Nagel et al., 1997).

Fig. 17.1. For the STCA, this value depends mostly on p_{noise}: Larger p_{noise} leads to smaller maximum flows. These maximum flow values, also called **capacities**, need to come out approximately correctly if one wants a model that is useful for reality. 2000 vehicles per hour and lane is a plausible value. Regional differences could be accomodated by different values of p_{noise}; this could even be made a function of the link. One however has to note that changes in p_{noise} also change the average acceleration of vehicles, which will, for example, change signal timing requirements or emissions. This is the reason why the CA approach can only be seen as a first, relatively rough starting point for a regional model. Once all other problems (such as demand generation) are sufficiently solved, the CA driving logic should be replaced by a model with continuous coordinates such as the ones discussed in Chap. **??[[maps]]**.

17.4 Lane changing

All lane changing rules, no matter if for CA or other models, follow a similar scheme (e.g. Sparmann, 1978): In order to change lanes, drivers need an incentive, and the lane change needs to be safe. An incentive can be that the other lane is faster, or that the driver eventually needs to make a turn. Safety implies that one needs enough space on the target lane. Thus, a simple lane changing condition can read as (Rickert et al., 1996a) (Fig. 17.2):

(I) Incentive: $min[v+1, v_{max}, gap_{other}] > \min[v+1, v_{max}, gap]$, i.e. the gap on the other lane is larger than the gap on the current lane, allowing a higher speed on the other lane.

Bounding the comparison at $\min[v+1, v_{max}]$ makes sure that only gaps sizes which are relevant for the car's current speed are considered.

(S) Safety: $gap_{other,back} > v_{back}$, i.e. the *backwards* gap on the other lane is large enough that a vehicle approaching with v_{back} does not have to slow down immediately.

Lane changing includes an additional sub-timestep, which is best exectued before the car following step. The full sequence is:

1. Go through whole system and tag vehicles for lane change.
2. Go through whole system and execute lane changes for tagged vehicles (sideways movement of vehicles).

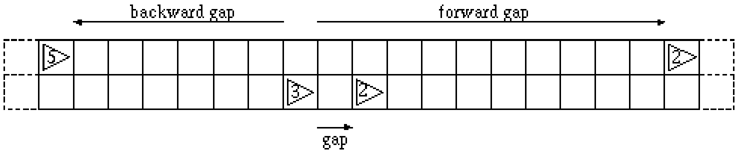


Figure 17.2: Lane changing. A smalle “gap” will give an incentive to change lanes. The lane change is actually executed if both “forward gap” and “backward gap” are large enough.

3. Go through whole system and compute new velocities.
4. Go through whole system and execute forward movement of vehicles.

The separation of the lane change into a tagging and a movement step is useful to maintain the parallel update: Because of reaction delays, driver decisions should be based on “old” information.

The above lane changing rules may have vehicles from both sides compete for the same cell in a middle lane. This can be overcome by making lane changes to the right only in even and lane changes to the left only in odd time steps. Another possible artifact are long rows of vehicles synchronously oscillating between left and right lane. This can be suppressed by executing the above lane changes with a probability smaller than one, for example 0.99.

All this together is essentially the lane changing criterion currently used in the Transims micro-simulation, and it seems to work reasonably well for U.S. traffic (Nagel et al., 1997).

The above lane changing criterion is symmetric, since changing to the left happens according to the same criterion as changing to the right. One result of this is that people stay in the left lane until some incentive pushes them out of it, again not totally unrealistic for traffic in the United States. For European (and other) countries, one has the constraint that passing on the right is not allowed, at least not when traffic is not congested. There are many ways to implement this. A fairly straightforward version is to change to the left when either on the same lane or on the left lane a slower vehicle is present:

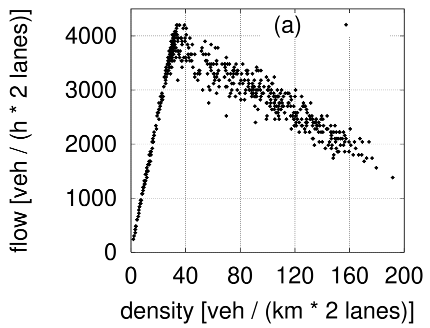
(I'.a) Incentive to go to left: “$v \geq v_r$ .OR. $v \geq v_l$” , where v_r refers to the vehicle in front on the same lane, and v_l refers to the vehicle in front one lane to the left.

Since the lane changing is no longer symmetric, many plausible rules are possible to trigger lane changes to the right. A good construction criterion for rules is to make lane changes to the right based on the logical negation of lane changes to the left. This results in

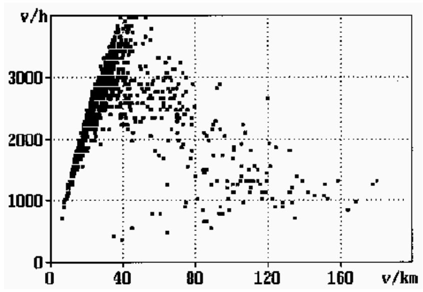
(I'.b) Incentive to go to right: “$v < v_r$ .AND. $v < v_l$”. Note that now v_l now refers to the same lane, and v_r refers to the lane to the right.

This leaves as a free parameter the distance d how far vehicles look forward for vehicles in the same and in the other lane. Larger d results in a stronger incentive to go to the left.

An important observation is that microscopic lane changing rules need not be realistic in order to generate plausible macroscopic traffic. For example, all lane changes according to the above rules happen in one simulation time step, which is usually one second, whereas in reality this takes longer (3–5 seconds). Also, the above rules result in too many lane changes when traffic on both lanes is similar – an effect that is annoying in animations (see, for example, one of the Transims videos), but macroscopic relations



[]



[]

Figure 17.3: Multi-lane fundamental diagrams. (a) STCA with $v_{max} = 5$, $p_{noise} = 0.25$. From Nagel et al. (1998). (b) Reality (Germany). From Wiedemann, published in Nagel et al. (1998).

such as fundamental diagrams still come out correct (Rickert et al., 1996a; Nagel et al., 1998).

As noted above, the incentive to change lanes could also come from an intended turn movement at the end of the link, and one can partially over-ride the safety criterion with increasing urgency of the incentive criterion.

17.5 Validation of lane changing rules

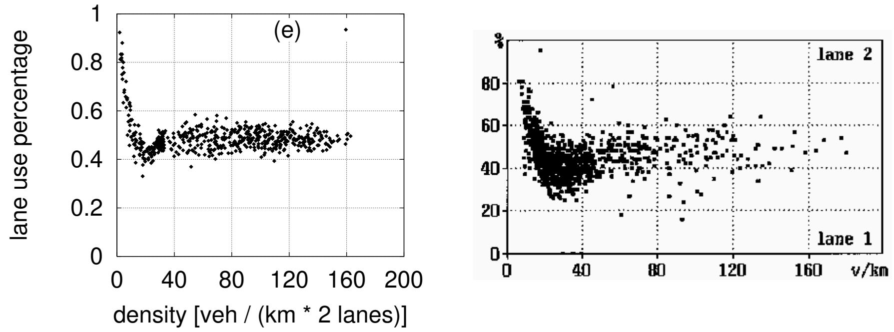
The most important issue for lane changing is that the fundamental diagram should remain plausible, i.e. with a maximum flow of about 2000 veh per hour and lane. This is indeed the case both with the above symmetric and the above asymmetric lane changing rules. A fundamental diagram for a simulation with asymmetric rules is in Fig. 17.5; compare this to a fundamental diagram from (German) reality in Fig. 17.5.

Another quantity of interest is the fraction of vehicles in each lane. For the symmetric rules and 2-lane traffic, this should always be at 50%. For the assymmetric lane changing rule introduced above, lane usage is plotted in Fig. 17.5, which was obtained with a look-ahead distance of $d = 16$ cells. Fig. 17.5 shows a plot of the same quantities from (German) reality. Additional rules, which can bring the simulations even closer to reality, are discussed by Nagel et al. (1998).

Another validation of lane changing rules concerns vehicles that change lanes in order to be in the correct lane for a turn. Two important questions here are how many vehicles do not reach their desired lane, and how much the lane changing disturbs the throughput. The first question is more critical under congested conditions, and one needs a set-up where the intersection capacity is smaller than the link capacity, caused for example by traffic lights. The second question is most critical near maximum flow; for example, one could test if at a traffic light just turned green, outflow is reduced when there is a lot of last-second lane changing.

17.6 Traffic signals

We now turn to intersections, where links, with car following and lane changing dynamics, are connected. The easiest case are fully signalized intersections since the signal (assuming it is working correctly) is taking care of avoiding crashes. The dynamics re-



[] []

Figure 17.4: Asymmetric lane usage. (a) Simulation. (b) Reality (Germany). **[[from wiedemann, published in ...]]**

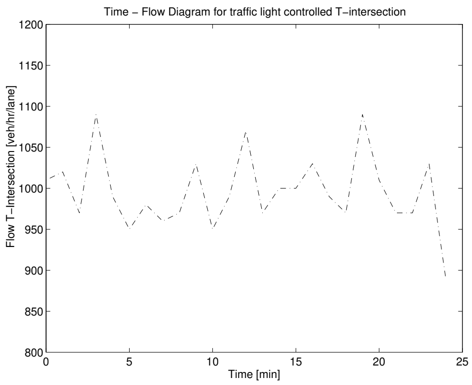


Figure 17.5: Number of vehicles going through the intersection per green phase, re-scaled to hourly flow rates per lane.

sulting from a red light can be generated by placing a virtual car with speed zero into the last spot on the link, and removing this car once lights turn green.

17.7 Validation of traffic signal rules

The most important quantity for traffic lights is the time headway between vehicles when the traffic light turns green. As a rough estimate, one can take the above-mentioned value of 2000 vehicles per hour and convert it into time headways, resulting in $3600/2000 = 1.8$ seconds per vehicle. More exact values need to be taken from local field data.

There is discussion if maximal flow on a freeway can be larger than the outflow from a queue, such as at a traffic light. For the STCA model that we are using so far, this issue is not critical; for other car following models it may play a role. More discussion of this is in Chap. 27.

17.8 Unprotected turns

Somewhat more difficult are unprotected turns, i.e. turns that are not regulated by traffic signals and where vehicles need to merge on their own without accidents. Typical

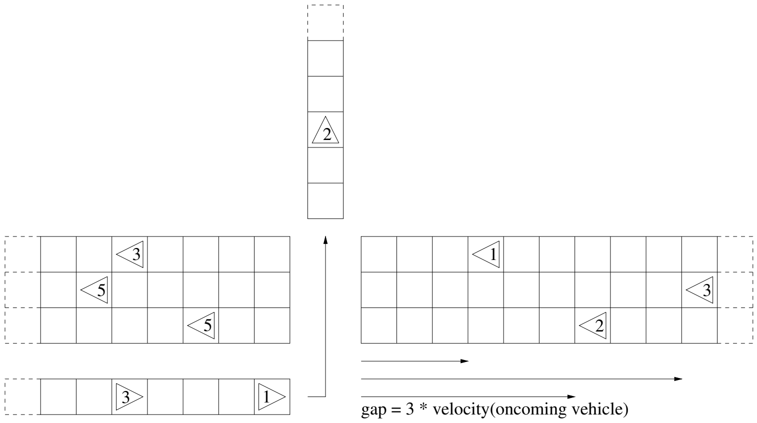


Figure 17.6: Illustration of gap acceptance for a left turn against oncoming traffic. From Nagel et al. (1997).

examples of this are yield, stop, "right on red", left turns against oncoming traffic, and on-ramps to freeways. The mechanism here is again a "gap acceptance" similar to the safety criterion (S) for lane changes (Fig. 17.6). That is, the vehicle on the incoming road moves into the major road if the gap there is big enough. This gap stretches upstream, since the incoming driver does not want the car upstream on the major road to crash into him/herself. The standard reference for highway engineers, the Highway Capacity Manual (Transportation Research Board, 1994a) states that drivers accept gaps that correspond to time headways of approximately 5 seconds or more, which means that the spatial gap needs to be proportional to the speed of the oncoming car (Fig. 17.6). In our standard CA implementation, this would mean that the accepted gap would have to be at least five times the oncoming vehicle's velocity. When implementing this rule, it turns out that a factor of three instead of five gives much more realistic flow rates (Nagel et al., 1997). It is not totally clear why this is the case.

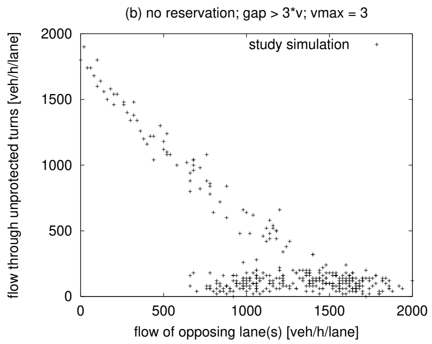
[[say something about merges/weaving. integration refs?]]

17.9 Validation of rules for unprotected turns

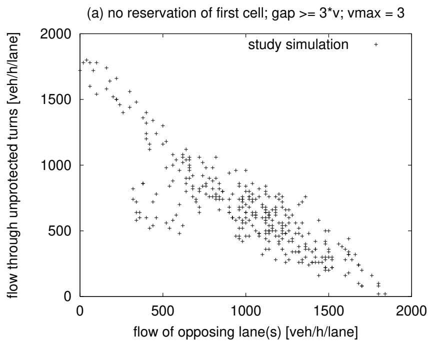
The typical measurement for unprotected turns is the maximum incoming flow rate as a function of the flow on the priority street. Such plots look like those in Fig. 32.6 with flow on the minor road (y-axis) as function of flow on the major road (x-axis). For interpretation, best start in the top left corner. Since there is no flow on the major road, flow from the minor road can enter at a high rate. With increasing flow on the major road, flow from the minor road is reduced. When the major road reaches capacity, the flow from the minor road is nearly zero. When the density on the major road goes above the maximum-flow density, then the flow on the major road is again reduced, but this time by congestion. In Fig. 17.9, vehicles from the minor road still have a hard time entering. In contrast, the gap acceptance rule from Fig. 17.9 allows vehicles from the minor road to enter into the major road under congested conditions, effectively modelling a "zipping" effect.

Two important messages are:

- Seemingly small changes, such as the change of gap acceptance from "$>$" to "$\geq$", can have large consequences. Such small changes can also easily be caused by the actual implementation of the rules. For example, in the Transims micro-simulation



[]



[]

Figure 17.7: Two different rules for the case of a 1-lane minor road controlled by a yield sign merging into a 1-lane major road. (a) Acceptance rule "accept if $gap > 3 \cdot v_{oncoming}$". $v_{max} = 3$. (b) Acceptance rule "accept if $gap \geq 3 \cdot v_{oncoming}$". Note that this seemingly small difference has a strong effect on throughput in the congested situation. (a) models that vehicles from the minor road cannot enter the major road once the major road is congested; (b) essentially models a "zipping" behavior, i.e. that vehicles from the major and the minor road alternate once the major road is congested.

traffic on the major street reserves cells on the outgoing link, even if in the end the vehicle does not claim it. This clearly reduces opportunities for vehicles from the minor road.

- Further details need to be taken from local conditions. For example, the flow from the minor into the major road when there is no traffic on the major road depends on speed limits and intersection layout, such as the curvature of the turn. This situation will rarely occur in reality, since if there is traffic on the minor road, there is usually also traffic on the major road. Exceptions are situations such as the end of soccer games or evacuation scenarios.

  Similarly, there are differences between yield and stop, and if the traffic from the minor street merges with the traffic from the major street, or crosses. Again, although the tendency of these changes are clear, exact flow values need to be taken from local conditions.

17.10 Discussion

In this chapter, we have further discussed improvements to the CA traffic simulation. It turns out that, for car traffic, such models consist of only four aspects:

- Car following
- Lane changing
- Protected turns
- Unprotected turns

Once these four aspects are implemented in a reasonable way, one has a basic model. From here on, considerable work is necessary to calibrate and validate individual details. In particular, lane changing needs to include lane changing to reach a particular lane for a turn, and lange changing on merge/acceleration lanes.

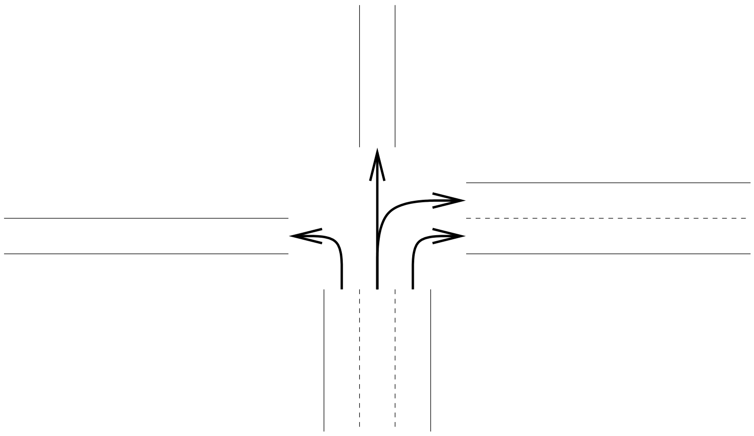


Figure 17.8: Lane connectivities across intersections. This information is needed for realistic multi-lane simulations.

A problem with such a microsimulation approach is that the necessary input data is often not available. For example, as a minimum one needs lane connectivities (which incoming lanes are connected to which outgoing lanes, Fig. 17.8), and signal plans. Furthermore, although it is an advantage that such simulations generate link capacity instead of taking it as input data, considerable adjustments need to be done. For example, the Gotthard tunnel, as a 1-lane road without traffic light, should have a capacity of 2000 vehs/hour. According to the local police, however, the capacity not more than half of that. The reason, presumably, is that the tunnel entrance has a strong uphill slope, and acceleration of vehicles is less than normal.

Chapter 18

The queue model for traffic dynamics

18.1 Introduction

In Chap. 7 we have introduced a simple cellular automata micro-simulation. The reason to chose that particular modelling technique was that it is conceptually simple, relatively easy to implement, somewhat realistic, and it fulfilled the functionality that was needed at that point in the project. In this chapter, an alternative will be presented, the so-called queue model (Gawron, 1998a). For experts: The queue model is essentially a standard queueing model, but with storage constraints added. Storage constraints mean that links can be full, which causes spillback across intersections.

The queue model is in our view the simplest dynamic model that is somewhat useful for real world predictions (see Chap. ??). Despite some obvious shortcomings in the description of the dynamics (see Chap. ??) in particular with respect to traffic jam wave backpropagation, we are not aware of any empirical evidence showing that more sophisticated models are truly better with respect to their predictive power. However, the path to more realistic simulations does not go via the queue model, but is a continuation of the explicit spatial methods, such as the CA. Making *those* methods, possibly on continuous rather than cellular space, useful for the real world (Chap. 17) is considerably more work than making the queue model useful for the real world. In consequence, if one intends to use the methods presented in this text for real world applications, one needs to carefully weigh advantages and disadvantages: The queue model of this Chapter is the fastest path to some usefulness, but is eventually limited; the CA model of Chaps. 7 and 17 (or non-cell based variants of this) are considerably more work but ultimately more realistic and more flexible.

18.2 General

From our general framework, we have the following requirements to a traffic simulation:

- Vehicles need to be able to follow plans. This implies that the simulation needs to be dynamic (i.e. time-dependent), and that some notion of individual vehicles needs to be present in the simulation.
- The simulation needs to be reasonably fast. A computational speed of at least 100 times faster than real time (i.e. simulating 24 hours of traffic in 0.24 hours of

computing time) is desirable in order to obtain bearable waiting times for the feedback/learning. This computing speed can be achieved by selecting small scenarios, by using simple models, or by parallel computing. This text concentrates on the last two aspects.

The important numbers characterizing a road from the perspective of transportation planning are:

- **Free speed**. This is the speed that vehicles drive on a link when no other constraints are present.
- **Flow capacity**. This is the maximum number of vehicles per time unit that can move over a link when no other constraints are present. In city traffic, the flow capacity is often determined by a traffic light at the end.
- **Storage constraint**. This is the maximum number of vehicles that can be on a link under jammed conditions.

The first two numbers are also used in all traditional transportation planning software (based on static assignment, see Chap. 28) and are therefore typically available with standard data files for transportation planning. The third number is necessary when a link is full and no more vehicles can enter, causing spillback. Without the storage constraint, flow demand above the flow capacity would allow an unlimited number of vehicles on the link, which is clearly not realistic.

The queue model bases its dynamics on free speed, flow capacity, and storage constraint only. Typical input data are, for each link a, the attributes free flow velocity $v_{0,a}$, length L_a, capacity C_a and number of lanes $n_{lanes,a}$. Free flow travel time is calculated by $T_{0,a} = L_a/v_{0,a}$. The storage constraint of a link is calculated as $N_{sites,a} = L_a \cdot n_{lanes,a}/\ell$, where ℓ is the space a single vehicle in the average occupies in a jam, which is the inverse of the jam density. One can use $\ell = 7.5$ m, as for the CA technique.

The arguably simplest intersection logic (Gawron, 1998b) is that all links are processed in arbitrary but fixed sequence, and a vehicle is moved to the next link if (1) it has arrived at the end of the link, (2) it can be moved according to capacity, and (3) there is space on the destination link (see Algorithm A in Fig. 18.1). More formally, the following happens:

- **Free speed:** A vehicle that enters link a at time t_0 cannot leave the link before time $t_0 + T_{0,a}$, where $T_{0,a}$ is the free speed link travel time as explained above.
- **Flow capacity:** The condition "vehicle can be moved according to capacity" is determined as
$$N < int(C_a) \text{ or } \Big(N = int(C_a) \text{ and } rnd < fr(C_a)\Big) \tag{18.1}$$
where $int(C_a)$ is the integer part of the capacity of the link (in vehicles per time step), $fr(C_a)$ is the fractional part of the capacity of the link, and N is the number of the vehicles which already left the same link in the same time step. rnd is a random number such that $0 \leq rnd < 1$. What it is meant by this formula is that the vehicles can leave the link if leaving capacity of the link has not been exceeded yet in this time step. If the capacity per time step is non-integer, then we move the last vehicle with a probability which is equal to the non-integer part of the capacity per time step.
- **"Space on destination link":** If the destination link is full, the vehicle will not move across the intersection.

18.3 Fair intersections

The queue model has the same problem as our simple CA model with respect to "fair" intersections (cf. Sec. 7.5). That problem is that the queue model dynamics as described so far goes through the links in a fixed order, meaning that some links always have the priority, and these may not be the links that should have the priority.[1]

A somewhat better way is to process the links in random order. We have already seen in Sec. 7.5 how to do this. Eventually however, one needs to introduce a proper intersection dynamics. A clean way to do this is the following:

1. **Move to a parallel update.** In a parallel update, all links are processed simultaneously. This means that all rules in order to move a configuration from time t to time $t+1$ can only depend on information from time t.

   For the queue model, this is achieved by remembering the number of empty cells on a link from time t. That is, if a link is full at time t, then no vehicles can enter during the update from t to $t+1$, even if the link opens up during that time step.

   A parallel update is also important in anticipation of parallel computing (Chap. 25).

2. **Separate link dynamics from intersection dynamics.**

   For the link dynamics, we introduce an additional buffer at the end of the link, as in Fig. 18.2. The size of the buffer is $\lceil C_a \rceil$, i.e. the smallest integer that is larger or equal to the capacity in "vehicles per time step". Vehicles are moved from the link proper into the buffer if the travel time constraint and the capacity constraint are fulfilled, and if the buffer has empty space. That is, this is exactly the same dynamics as before, except that we move vehicles into the buffer instead of across the intersection. – This update is done by iterating over all links.

   For the intersection dynamics, an additional loop is introduced, which is over all nodes. Here, vehicles are moved from the (incoming) buffers to the outgoing links. Neither travel time nor capacity constraints need to be considered here because they were already treated before.

   This approach is borrowed from lattice gas automata, where particle movements are also separated into a "propagate" and a "scatter" step (Frisch et al., 1986).

When looking to our framework from Sec. 7.7, one notices that we have already the provisions for separating link dynamics from intersection dynamics: there are already two loops, one going over all links and the other over all nodes/intersections.

Regarding the intersection dynamics for the queue model, many solutions are possible. For example, it is possible to go through the incoming links in random order weighted by capacity, thus giving a higher priority to links with high capacity. Again, there are several ways to do this, for example to re-select the link for each vehicle to move until all moves are exhausted, or to process one link until its moves are exhausted and only then move to the next link. Although none of these are difficult to implement, there are subtle differences between them when used for complicated intersections. A possible algorithm is given as Algorithm B in Fig. 18.3.

[1] Note that the winning links are not the ones that come first, but the ones that come first after the outgoing link was treated. For example, assume a configuration where links 1 and 3 are incoming into link 2, and assume that they are processed in sequence 1, 2, 3. **[[fig?]]** Also assume that under congested conditions initially all links are completely full. Then link 1 is processed first, but link 2 is full, so no vehicle can move. Then link 2 is processed, and some vehicles move out, opening up some space. Finally, link 3 is processed, and since there is some space on link 2, some vehicles can move.

```
for all links do
  while vehicle has arrived at end of link
  AND vehicle can be moved according to capacity
  AND there is space on destination link do
    move vehicle to next link
  end while
end for
```

Figure 18.1: Algorithm A – Arguably simplest intersection algorithm

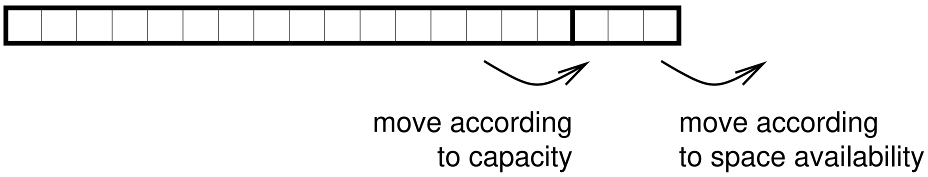


Figure 18.2: The separation of flow capacity from intersection dynamics.

18.4 Limitations of the queue model

In the introduction to this chapter, it was pointed out that the queue simulation is eventually limited in terms of its realism. In this section, these limitations will be discussed.

A first limitation concerns the dynamics of traffic jams. In the queue model, when a vehicle leaves a link, that free spot becomes available for entering vehicles very quickly: In Algorithm A, it becomes available immediately; in Algorithm B, it is somewhat delayed by the buffer dynamics and the parallel update. In both cases, however, the time that it takes until it becomes available for entering vehicles *does not depend on the link length*. This is in stark contrast to reality, where such "holes" travel with a finite speed of approximately $15\ km/h$. The reason for the real-world behavior becomes immediately obvious if one looks at the corresponding dynamics in the CA, where a hole in a completely dense jam is slowly passed on against the traffic direction by at most one vehicle movement in each time step; this is discussed in more detail in Chap. 27.

This limited realism in terms of traffic jam dynamics shows up when solid jams in the queue model, for example caused by an accident, are dissolved: Instead of being dissolved at the downstream end only, such jams in the queue model are dissolved quasi-simultaneously along the whole length. **[[fig portland]]** It seems however that this problem can be resolved via additional rules, such as a limitation on the "speed of holes" (**?**).

Other limitations are concerned with the limited vehicular and spatial resolution:

- **Interaction between slow and fast vehicles.** On multi-lane roads, fast cars can pass slow cars as long as traffic is light. Only when traffic becomes denser, then fast cars are caught between slow cars. In the queue simulation, all cars are assumed to drive with the same speed.

- **Interaction between different vehicle types.** Examples for this are interactions between pedestrians and cars, bicycles and cars, or between buses/light rail and cars.

- **Signal phases.** Diligent signal phasing can make an enormous difference to an intersection capacity. This cannot be captured by simple intersection capacities, since it depends on how traffic streams and signal phases work together.

```
// PROPAGATE VEHICLES ALONG LINKS:
for all links do
  while vehicle has arrived at end of link
  AND vehicle can be moved according to capacity
  AND there is space in the buffer (see Fig below) do
    move vehicle from link to buffer
  end while
end for
// MOVE VEHICLES ACROSS INTERSECTIONS:
for all nodes do
  Mark all links that are incoming to this node
  while there are marked links do
    Select a marked link randomly proportional to capacity
    Un-mark link
    while there are vehicles in the buffer of that link do
      Check the first vehicle in the buffer of the link
      if its destination link has space then
        Move vehicle from buffer to destination link
      end if
    end while
  end while
end for
```

Figure 18.3: Algorithm B – Links and Intersections separated

- **Complicated street layouts.** Merging, turning, and weaving lanes make a substantial difference to traffic flow. Most importantly, turning lanes, i.e. the separation of vehicle streams by turning direction, prevents situations such as in Fig. 18.4, where a left turning vehicle blocks all the traffic behind it. This becomes particular important in conjunction with signal phases, since optimally the turning lanes are emptied out during each green phase. That is, turning lanes of the correct length ensure that the green phases of an intersection are used optimally.

- **Weaving,** in particular if large numbers of vehicles enter a street on the right lane(s) but want to exit it on the left lane(s).

For such effects, the simple queue simulation is no longer sufficient. Sometimes, parameterizations of certain effects are available, but in general it will be necessary to resort to a more realistic type of micro-simulation. In such a more realistic micro-simulation, one will not only have individual cars with different individual characteristics, but also realistic street layouts, signals, bicycles, pedestrians, light rail and buses, etc.

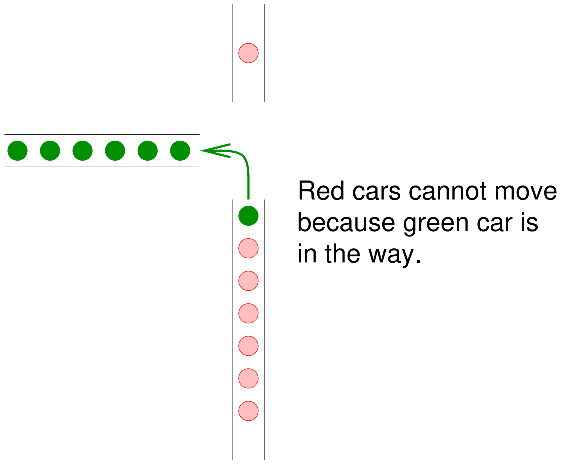


Figure 18.4: Problem of FIFO-based models

Chapter 19

Routing

[[get some papers]]

[[Cascetti??]]

19.1 Time aggregation

19.2 Generalized cost functions

19.3 Alternative routes

In our approach, each new route was generated as what would have been the fastest route on the previous iteration.[1] It is improbable that real people solve this problem exactly, and for that reason alternative route generation algorithms are desirable. Somewhat interestingly, it turns out that finding alternative routes is considerably more difficult than finding the fastest path alone.

One option is to systematically compute the second-fastest, third-fastest, ..., k-fastest path. This is however much more compute-intensive than computing the shortest path alone (Yen, 1971; Perko, 1986; Clarke et al., 1963; Chabini, 1998a). In addition, most of these paths are not plausible for the real world. Often, they are just small variations of already existing paths, with for example leaving the freeway and returning to it at the same entry/exit point. Only very few of the paths generated in this way are true innovations.

As an alternative, one could attempt to generate routes heuristically, instead of systematically. This is also not a simple problem (Park and Rilett, 1997). Typical heuristic approaches start searching in the geographic direction of the destination, and in consequence often miss freeway connections which demand some backtracking in order to reach them. More sophisticated approaches will be necessary here.

One may think that heuristic approaches might also be desirable for computational speed reasons in very large road networks. In practice, we have never found this to be a problem. In a typical transportation planning network, with a size of about 10 000 nodes and 20 000 links, a straightforward implementation of the time-dependent Dijkstra algorithm allows the computation of 10 000 new routes per second on a typicaly 500 MHz CPU (Jacob et al., 1999), which is fast enough for practical cases. In much larger networks,

[1] To be entirely precise, one would have to say that the route is best based on the time-averaged information that the router uses.

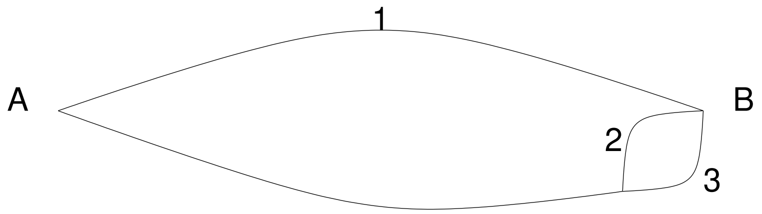


Figure 19.1: Correlations between paths

this may no longer be sufficient. In such cases, some hierarchical pre-processing can help. This is a topic of ongoing research.

19.4 Logit for routes

Another major problem of our approach is that all travellers with the same situation will be put on the same route, that is, there is no "spread" of solutions.

A typical way to obtain some spread of solutions is to use a logit approach. Remember, a logit means that the probability of picking a solution i is set to

$$p_i = \frac{e^{\beta U_i}}{\sum_j e^{\beta U_j}} , \tag{19.1}$$

where U_i is the utility of solution i. When the utility of a solution is high, then it will be selected with a high probability.

For routes, utility is negative, and it becomes more negative the longer the driving time. For example, one could set $U_j = -T_j$, where T_j is the driving time for route choice j.

A major problem with this is that it is not easy to generate routing alternatives. Two approaches, and their drawbacks, are:

- It is possible to compute k-shortest paths.

  Then, it is problematic to use logit on routes (e.g. (Cascetta and Papola, 1998)). This is actually easy to see: In Fig. 19.1, there are three paths from A to B. Assume they have all the same travel time. The plausible solution then is that path 1 is used with probability 0.5, and paths 2 and 3 are used with probability 0.25 each.

  The logit solution will however be that all three paths are used with equal probabilities $1/3$.

  The example can be made arbitrarily pathologic by adding more "short" alternatives.

  It is however possible to use more sophisticated models than the logit models (Cascetta and Papola, 1998).

- Another method is to only generate routes which are "real" alternatives (Park and Rilett, 1997). This is however not an easy problem in itself.

  And the problem with the logit still applies, although to a weaker extent.

19.5 Planning for given arrival time

[[todo]]

19.6 Mental maps

[[todo]]

Chapter 20

Non-car modes of transportation

20.1 Routing

Another problem is how to include public transportation. It is possible to do this in the router, that is, the router should figure out if, say, public transportation or car is a better route for a certain trip (Barrett et al., 2000).

An alternative is to include the mode choice into the activities generation, i.e. where we have adjusted the trip starting time in the past.

20.2 Simulation

Realistic micro-simulations also need to simulate other modes of transportation besides the car, such as buses, light rail, walking, bicycle. This makes micro-simulation codes considerably more complicated to program and to run, the latter in particular since all the additional information needs to be coded into file, which need to be interpreted correctly by the simulation.

There is however a trick which considerably simplifies the situation in many cases: As long as there is no congestion and no interaction between modes, modes can be treated as "following there schedule". That is, without congestion a subway or a bus will just depart and arrive as noted in the schedule, and a pedestrian will walk exactly with the expected speed. Since this means predictable behavior, such trips or legs can be preplanned by the router, and the microsimulation just follows the plan. More technically, if a car-only microsimulation encounters a leg which is not car-based, it would process the leg according to departure and arrival information from the plan. In this way, the problem of multi-modal traffic is delegated to the router.

The situation changes when the other modes suffer from congestion, or when there is interaction between modes. Examples of the former are pedestrian congestion in subway stations, or overcrowded buses. An example of the latter is the interaction between pedestrians and cars on crosswalks. In those cases, a direct implementation of other modes into the micro-simulation will be necessary. Some elements, such as buses or light rail stuck in traffic, can be modeled within the queue model. For other aspects, more realistic micro-simulations will be necessary.

In such a more realistic micro-simulation, some aspects can in fact be modeled without too much effort. For example, buses are treated similarly to cars (i.e. they follow a route), with the distinction that every time they approach a bus stop, they move into the right lane and stop there. A light rail ("Tram") is modelled essentially a bus but with very strong lane restrictions, that is, it has to stay on its tracks. If the tracks are embedded in regular traffic, then the tram will just do standard car following; if the tracks are separate, then the tram will run at free speed except for stops.

Other interactions are more difficult to model and need additional or separate models. For example, pedestrian congestion follows different rules than traffic congestion; there are computer codes which simulate this. One could connect such a pedestrian code with a traffic simulation code. Major implementation problems occur when such simulations need to be coupled, for example, when pedestrians crossing a street interact with the car traffic on the street. Little technology seems to be known to couple these simulations without having to rewrite at least one of them to integrate it into the code of the other. Our own expectation is that for the foreseeable future enough progress can be made by working on other aspects of the problem, until some better technology becomes available. Clearly, other areas of simulation have similar problems.

Chapter 21

Demand

Once the synthetic population is generated, all other modules act directly on the agents. What is necessary here is a procedure that as a result generates travel demand, i.e. the wish of people to move from one location to another. As already said in 2.2, **[[check]]** two important methods here are: (i) origin-destination matrices, and (ii) activity-based demand modeling.

21.1 Origin-destination matrices

As also already said in Sec. 2.2, 2.2, **[[check]]** origin-destination (OD) matrices contain the number of trips from n starting points to n destinations; it is therefore an $n \times n$ matrix. As also said, these matrices can refer to arbitrary time periods; these days, one typically uses "morning peak" and "afternoon peak" periods.

There are many ways to obtain origin-destination matrices. In transportation planning, the typical methods is to anchor them to the land use, and to use behavioral "rates" to determine trip frequencies (e.g. (Lohse, 1997)). Residential areas "produce" so and so many trips per capita; commercial areas "attract" so and so many trips per capita. The matching of origins to destinations is done via gravity methods, i.e. the probability of a trip to go to a certain destination is some function of the attraction of this destination and the generalized cost of getting there.

Another method is to derive OD matrices from traffic counts. Here, one collects counts on as many links of the transportation network as possible, and then uses statistical estimators to derive OD matrices from this (e.g. (Cascetta et al., 1993)). Statistical estimators are necessary because the problem is under-determined. Sometimes, the two approaches are combined, i.e. the historical OD-matrices are used as starting points, but they are corrected via traffic counts (DYNAMIT, accessed 2003).

21.2 Activities-based demand modeling

The problem with OD matrices is that they fix the travel demand once they have been derived. Thus, they fail to generate the effect of "induced" travel, which usually happens when one expands capacity. For example, a new freeway may induce people to make more trips, thus increasing overall travel. This means that one needs a demand generation method that is elastic with changing supply.

Activity-based methods attempt to achieve this by generating directly what people do during a day and where; transportation demand is thus derived by connecting activities

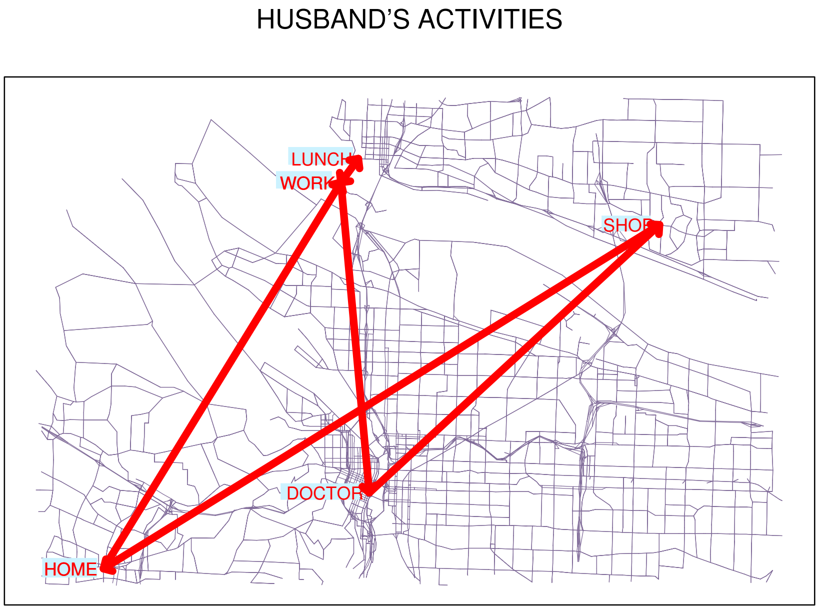


Figure 21.1: Example of a sequence of activities for a person in Portland/Oregon. From R.J. Beckman.

at different locations (Fig. 21.1). There are at least two different methods to generate activities: econometric, and heuristic.

In principle, one can derive OD-matrices from activities, and many groups do this because it connects activity-based demand generation to existing models. This has, however, to be done with care since one loses important information. An important example of lost information are trip *chains*, where a person may go to work, may go shopping, and then home. If the person gets stuck on the way to shopping, the trip from shopping to home will take place later than anticipated; such effects do not get picked up in the OD matrix. Also, a universal reaction to changes in congestion seems to be to add or suppress intermediate stops at home, i.e. to replace home-work-home-shop-home by home-work-shop-home or vice versa. One would have to be careful to not suppress these possibilities when translating the trip chains into OD-matrices.

Econometric **[[I have discrete choice theory now in "background"]]**

[[dennoch k"onnte man fast alles hier lassen]]

[[need to sort out the β_i]]

Econometric methods (Ben-Akiva and Lerman, 1985; Domencich and McFadden, 1975) are based on random utility theory, which will be explained in more detail in Chap. 29. An often-used choice model is the so-called logit model. If there are several options $i = 1..N$, then the logit model predicts that the probability to select option i is

$$p_{a,i} = \frac{e^{\beta \, V_{a,i}}}{\sum_j e^{\beta \, V_{a,j}}} \,, \tag{21.1}$$

where $V_{a,i}$ is the utility ("score") of option i for a particular individual a, and β is a parameter characterizing randomness. This equation was already used in Sec. 14.3, and the consequence of varying β was discussed there.

[[have used U in dep time choice. should use same notation as ben-akiva]]

For demand generation, one needs to make $V_{a,i}$ dependent on the attributes of the options, and on the properties of the individual under consideration. A typical assumption is to make this dependence linear:

$$V_i = \beta_1 \, x_{a,1} + \dots + \beta_k \, x_{a,k} + \beta_{k+1} \, x_{i,k+1} + \dots \, , \tag{21.2}$$

[[now I have used β twice, in slightly different meanings.]]

where the $x_{a,j}, j \leq k$ are person attributes, and the $x_{i,j}, j > k$ are option attributes. For example, one could have

[[find one with bus, car, income]]

Utility theory assumes that the utility a person i sees in a certain action a is composed of a measurable and a non-measurable part:

$$U(i,a) = V(i,a) + \eta(i,a) \, . \tag{21.3}$$

Under a variety of assumptions, e.g. that η is a random variable and follows a certain distribution, this leads to an equation for the probability to choose action a.

An often-used discrete choice model is the so-called logit model. Its main assumptions are:

- Individuals and actions are characterized by certain attributes, that is, two individuals with the same attributes will be modeled by the same equation. This also means that i and a are replaced by a vector of attributes, $\mathbf{x}_{i,a}$.

- The measurable part of the utilities, V, is a linear function of the attributes, i.e. $V = \beta \cdot \mathbf{x}$.

- The random variables η do not depend on the attributes $\mathbf{x}_{i,a}$, and they are Gumbel distributed, i.e. the generating function is

  $$F(\eta) = \exp[-e^{-\mu\,(\eta-\gamma)}] \, , \tag{21.4}$$

  which results in the distribution

  $$f(\eta) = \mu \, e^{-\mu\,(\eta-\gamma)} \, \exp[-e^{-\mu\,(\eta-\gamma)}] \tag{21.5}$$

  γ is a location parameter, and μ is a positive scale parameter. This distribution is somewhat similar to an asymmetric version of the normal distribution; its main advantage is that it leads to a closed form solution of the choice model.

With a logit model, the probability to choose the bus in a decision between bus and car could look as follows:

$$P(bus) = \frac{\exp[-\beta_b \, t_b]}{\exp[-\beta_b \, t_b] + \exp[-\beta_c \, t_c]} \, . \tag{21.6}$$

t_b and t_c are the respective travel times the trip would take by bus or by car. β_b and β_c are factors which weigh time in the bus vs. time in the car, i.e. they are "values of time". For example, one could say that time in the bus is more productive than in the car because one can read, resulting in $\beta_b > \beta_c$. However, usually the car is faster, compensating for this effect. – Note that Eq. 21.6 has the same functional form as a Boltzmann distribution.

The β_b and β_c are estimated from surveys, for example via maximum likelihood methods. A sample of the population with different car and bus travel times is asked about their choices, and the β_x are determined such that the probability according to Eq. 21.6 to re-generate the survey is maximized.

For applications inside a transportation simulation, this becomes a lot more complicated. An implementation for Portland/Oregon (Bowman, 1998) determines activity patterns (for example home-work-home or home-work-shop-home), activity timing, activity locations, mode choice, etc. As long as one wants to treat all alternatives simultaneously, this has the problem that the number of coefficients grows exponentially. For example, if one has five activities patterns, and three modes of transportation, this means 15 different choices and thus 15 parameters. If however one does not treat the alternatives simultaneously, one can make mistakes: For example, a person could have a strong preference for a pattern home-work-home-shop-home when averaged over *all* possible circumstances, but may prefer home-work-shop-home when really good bus service is available. When choosing first the pattern and then the transportation mode, this information gets misrepresented.

Heuristic methods The econometric method has a solid theoretical foundation, and it is currently the only method that is functional for transportation simulations. However, sometimes it seems like it does not really represent how people behave. The discrete choice method pretends that people calculate utilities for all possible alternatives and then choose the alternative with the highest utility. (Remember that the randomization just comes in because of "unobserved attributes".) However, people do not do this. For example, they may discard an activity pattern home-shop-work-home right away without calculating the utilities of all possible constellations.

Heuristic methods attempt to better represent such human planning processes. For example, research shows that humans make their planning decisions on many time scales simultaneously (Doherty and Axhausen, 1998). The time for work is usually alloted way in advance, shopping may be planned a day in advance, and then the whole schedule may be changed short-term because the child gets sick. Prototypes for such models exist, but they seem currently far away from being operational in any meaningful way.

It should be noted that heuristic and econometric methods can be combined. For example, one could use a heuristic method to determine which decisions are made how far in advance, and use an econometric method to make the actual decision. Or the econometric method could calculate the probability for each activities pattern, the heuristic method could decide to retain the two most important patterns, the econometric method than could calculate the utilities for these two patterns for all mode and time combinations, etc.

Summary of activities-based methods Activities-based demand generation models are a promising method for transportation simulation. Some implementations of these methods have reached the state where they can be used for actual applications (Bradley, 1997). However, so far there are only very few results about coupling these methods together with transportation micro-simulations, as intended with the transportation planning simulation packages described in this article. The only functional system that we are aware of uses a very simple method of demand generation; it is described in the appendix. But we are optimistic that research in the next couple of years will expand the boundaries in these areas enormously.

Chapter 22

Feedback

22.1 Introduction

A major shortcoming of the departure time choice of Chap. 14 is that the trip time is treated as being independent from the starting time. This is obviously not realistic. There are many ways to improve this. Two possibilities are described in the following. In addition, the difference between day-to-day and within-day replanning is shortly discussed.

22.2 Global trip times table

[[I may have this now in the do-it-yourself part.]]

Recall that the missing information is the expected trip time for a given starting time. One option is to generate a global trip times table, i.e. for each time slice and each origin-destination pair the information about the trip time for a departure time within that time slice. This table would be generated from actual performance of simulated travelers/vehicles, that is, all travelers/vehicles departing during the time slice from the same starting location to the same destination would be included, for example by averaging. The table would then be used by the activities generation module to provide estimated trip time information.

The main disadvantages of this approach are:

- In a large network, there are easily several hundred thousand links, corresponding to several hundred thousand potential origins/destinations. That is, for a single time slice, our table would have more than $10^5 \times 10^5 = 10^{10}$ entries, corresponding to 40 GByte per time slice, which is clearly too much for most current computing environments.
- Going along with the last is that in such a network, with a realistic number of 10^7 travelers, most entries of the trip time table would be left empty, implying some other method to fill the missing cells.

Implementation

For our simulations, this could be implemented as follows:

From the events file, generate a table of 5min-by-5min origin-destination trip times. That is, for each origin-destination pair and for each 5min bin, you average the travel times of vehicles during that 5min bin.

For example, if there were, between 8:00 and 8:05 (planned departure times), two vehicles traveling from link 100 to link 1900, and the trip took them 30 and 32minutes, respectively, then the expected trip time for a departure between 8:00 and 8:05 is 31 minutes.

Generating this table would concern the system integration specialists.

That table now is read into the activities generation module, and the departure time choice is based on that information.

This would concern the route/acts gen specialists.

If there is information missing between time bins, then interpolate. If there is information missing for early or late times, think about some intelligent solution.

22.3 Agent data base

An approach which seems in general much more robust is the use of an agent database. Here, we mean that each traveler/agent keeps a memory of options that he/she tried out, and some measure of the performance of each option. This approach is similar to classifier systems, genetic algorithms, or reinforcement learning, with the difference that the number of agents, typically several millions, is much higher in large scale transportation simulations than in typical applications of the mentioned areas.

The simulation would start with each agent having one or more options, which all have preliminary scores. Each iteration would consist of the following steps:

- Each agent would chose an option according to the scores, for example taking the option with the best score.
- The simulation would be carried out.
- Each agent would note the new score of the option that it just carried out.

In addition, it is necessary to inject new options into the system. For example, in each iteration one could give new options to a fraction of the agents, and then “force” those agents to immediately try them out. If these options lead to bad scores, the agents will rarely or never try them again.

Although such an approach is easy to state in principle, it is difficult to implement in practice because of performance limitations. Using a relational database such as MySQL is possible but slow with several millions of agents. Also, although a relational database provides support such as indexing and sorting, it’s emphasis is on consistent and secure operation, not on computational speed. This is a subject of active research.

Implementation

With respect to our practical examples, the easiest solution is to not worry about the routing choice, but remember starting times and performance only. That is, after a simulation run one would parse the events file, and for each agent note the starting time and the corresponding trip time. That information would be merged together with pre-existing information into some agent data base.

(One could for example do a flat file of agent performance for each iteration; the departure time choice module would then read *all* these files.)

For each agent that does departure time choice, the experienced trip times would be used as a base. For departure times outside the experienced interval, free speed travel times could be used. For departure times in between experienced travel times, some kind of interpolation (e.g. linear) could be used.

Note that agent memory needs to age, otherwise agents may remember information that is no longer relevant. One possibility would be to only read the agent experience from the last 10 iterations.

This would again be a cooperation between the systems integration specialists and the route/acts gen specialists.

22.4 Day-to-day vs. within-day re-planning

Day-to-day replanning assumes, in a sense, "dumb" particles. Particles follow routes, but the routes are pre-computed, and once the simulation is started, they cannot be changed, for example to adapt to unexpected congestion and/or a traffic accident. In other words, the strategic part of the intelligence of the agents is external to the micro-simulation. In that sense, such micro-simulations can still be seen as, albeit much more sophisticated, version of the link cost function $c_a(x_a)$ from static assignment, now extended by influences from other links and made dynamic throughout time. And indeed, many dynamic traffic assignment (DTA) systems work exactly in that way (e.g. (Bottom, 2000)). In terms of game theory, this means that we only allow unconditional strategies, i.e. strategies which cannot branch during the game depending on the circumstances.

Another way to look at this is to say that one assumes that the emergent properties of the interaction have a "slowly varying dynamics", meaning that one can, for example, consider congestion as relatively fixed from one day to the next. This is maybe realistic under some conditions, such as commuter traffic, but clearly not for many other conditions, such as accidents, adaptive traffic management, impulsive behavior, stochastic dynamics in general, etc. It is therefore necessary that agents are adaptive (intelligent) also on short time scales not only with respect to lane changing, but also with respect to routes and activities. It is clear that this can be done in principle, and the importance of it for fast relaxation (Esser, 1998a; Rickert, 1998) and for the realistic modeling of certain aspects of human behavior (Axhausen, 1990; Doherty and Axhausen, 1998) has been pointed out.

Chapter 23

Other Modules

freight

emissions

housing

land use

Chapter 24

Better file formats

24.1 Introduction

In the longer run, the file formats used in the "do-it-yourself" part are not very robust. The main problem is that with each change of the file format, several pieces of the simulation package need to be adapted consistently. Two ways to improve the situation are (a) use the header line not just for consistency checking, but to obtain the information of the content of each column; (b) use XML (extended markup language). This will be described in the following.

24.2 Use header line

In the "do-it-yourself" part, the header line was only used for consistency checking, for example for the nodes file

```
// process header line:
for ( int ii=1; ii<=NTOKENS; ++ii ) {
    inFile >> aString ;
    switch( ii ) {
    case 1: assert( aString=="ID" ) ; break ;
    case 2: assert( aString=="EASTING" ) ; break ;
    case 3: assert( aString=="NORTHING" ) ; break ;
    }
}
```

A more robust alternative would be to use the header line as an indication of what each column contains. Processing of the header line would essentially become

```
// process header line:
for ( int ii=1; ii<=NTOKENS; ++ii ) {
    inFile >> aString ;
    if ( aString=="ID" ) {
        column_id=ii ;
    } else if ( aString=="EASTING" ) {
        column_east=ii ;
    ...
    }
}
```

These columns would later be used during the file reading, for example via

```
// main loop:
while( !inFile.eof() ) {
    ...
    for ( int ii=1; ii<=NTOKENS ; ii++ ) {
        if ( ii==column_id ) {
            inFile >> nodeId ;
```

```
            } else ( ii==column_east ) {
              inFile >> xCoord ;
            ...
            }
        }
    }
```

This is in fact not much more work to program, and considerably more robust. The main reason why it was not introduced ealier is that it does not solve one of the main inconveniences, which is the parsing of the route-plans file. The problem with route-plans is that they are not column-oriented, and they cannot be, since the number of nodes in a route is changing from one route to the next. The next section discusses a robust way out of this dilemma.

24.3 XML

XML (extendsible markup language) is a system to describe unstructured data for computers. The main idea is that each item of the data is described *right where it shows up* instead of somewhere else in the file or even outside it. An XML nodes file would look like

```
<nodes>
<node id="15" x="123.45" y="678.9" />
...
</nodes>
```

That is, the information of where the id or the x/y coordinates are is repeated for each entry. This makes for larger files and slower parsing speeds, but the disadvantages are not that big:

- Since this is a standardized method, fast parsers are available.
- The overhead is not more than a factor of two.
- If keywords are repeated often (as they are for our files), compression tools will find that out so that compressed XML files are not much larger than compressed files without XML tags.

In general, parsers of XML files will not break when the input format is extended. For example, when additional keyword-value-pairs are added, they will just be ignored.

The main advantage of XML files is for the description of travelers' plans, where one now does not need all those awkward conventions any more. A route-plans file will for example look like

```
...
<person id="34">
<trip starttime="8h03" dplink="123" arlink="456" eta="8h33">
<nodes> 23 34 63 62 24 </nodes>
</trip>
</person>
...
```

This describes a trip from link 123 to link 456, with a starting time at 8h03, and an estimated arrival time at 8h33.

Further information, such as deomgraphic data or activities, can now just be added to the same file structure, e.g.

```
...
<person id="34" income="10000">
<act type="h" link="123" etime="8h03" />
<trip mode="car" starttime="8h03" dplink="123" arlink="456" eta="8h33" >
<nodes> 23 34 63 62 24 </nodes>
</trip>
```

```
<act type="w" link="456" duration="8h" />
<trip mode="car" starttime="16h33" dplink="123" arlink="456" eta="17h00">
<nodes> 24 62 63 34 23 </nodes>
</trip>
<act type="h" link="123" />
</person>
...
```

This would describe a person with id 34 and an income of 10000, which, at the beginning of the simulation, is doing at “at-home” activity, at link 123. At 8h03, the person starts driving to work, where she expects to be at 8h33. The person works for 8 hours, and then drives back home.

This is in principle a very flexible concept. In particular, there are no longer different files for activities, trip requests, (route-)plans, etc; everything is just one file format. For example, the router request (formerly “trips file”) would just be

```
...
<person id="34" income="10000">
<act type="h" link="123" etime="8h03" />
<trip mode="car" dplink="123" arlink="456"/>
<act type="w" link="456" duration="8h" />
<trip mode="car" dplink="123" arlink="456"/>
<act type="h" link="123" />
</person>
...
```

and the router would calculate all trip starting times, estimated arrival times, and sequences of routes.

As an alternative, there could be separate scheduling and routing modules.

The main issue here is that there is absolutely no standardization available yet. It is neither clear which concepts are simple in terms of modeling and simulation, nor which concepts are faithful in terms of human behavior. We will return to some of the latter in Chap. ??. **[[check ... doherty acts scheduling]]**

24.4 Some discussion

Why has the do-it-yourself package of this text not used XML? The main problem is that the parsers are not yet standardized. For example, for unix the C++ computer by itself is no longer sufficient; one needs to add some additional software. We expect the situation to be similar under other operating systems. In addition, the situation with parsers still is in a state of flux. That is, a parser that works today may not work any longer is a couple of months from now. For all other pieces of our package, we expect that it will work on standard systems for many years into the future.

For all those reasons, *this* text does not use XML files, but standard text files. However, there is a public domain version of our work, currently at **[[where]]**, which uses XML and which can be used as a starting point for further development.

Chapter 25

Parallel computing

25.1 Introduction

As we have seen, the computational requirements for a large scale simulation can be rather large, and eventually waiting for a result can take too much time. Using parallel computers is a way to improve the situation. When done right, using 100 parallel computers can reduce the waiting time by a factor of 100, for example from 100 days to one. Aspects of this are described in the following.

Note: The following still refers to cellular automata simulation methods. The spirit of the results is however also valid for the queue simulation used in the class.

[[the following (commented out) needs to be adapted/included/sorted]]

25.2 Micro-simulation parallelization: Domain decomposition

An important advantage of the CA is that it helps with the design of a parallel and local simulation update, that is, the state at time step $t + 1$ depends only on information from time step t, and only from neighboring cells. (To be completely correct, one would have to consider our sub-time-steps.) This means that domain decomposition for parallelization is straightforward, since one can communicate the boundaries for time step t, then locally on each CPU perform the update from t to $t + 1$, and then exchange boundary information again.

Domain decomposition means that the geographical region is decomposed into several domains of similar size (Fig. 25.1), and each CPU of the parallel computer computes the simulation dynamics for one of these domains. Traffic simulations fulfill two conditions which make this approach efficient:

- Domains of similar size: The street network can be partitioned into domains of similar size. A realistic measure for size is the accumulated length of all streets associated with a domain.
- Short-range interactions: For driving decisions, the distance of interactions between drivers is limited. In our CA implementation, on links all of the Transims-1999 rule sets have an interaction range of 37.5 meters ($=$ 5 cells) which is small with respect to the average link length. Therefore, the network easily decomposes into independent components.

We decided to cut the street network in the middle of links rather than at intersections (Fig. 25.2); THOREAU does the same (Niedringhaus et al., 1994). This separates the traffic complexity at the intersections from the complexity caused by the parallelization and makes optimization of computational speed easier.

In the implementation, each divided link is fully represented in both CPUs. Each CPU is responsible for one half of the link. In order to maintain consistency between CPUs, the CPUs send information about the first five cells of "their" half of the link to the other CPU. Five cells is the interaction range of all CA driving rules on a link. By doing this, the other CPU knows enough about what is happening on the other half of the link in order to compute consistent traffic.

The resulting simplified update sequence on the split links is as follows (Fig. 25.3):[1]

- Change lanes.
- Exchange boundary information.
- Calculate speed and move vehicles forward.
- Exchange boundary information.

The Transims1999 microsimulation also includes vehicles that enter the simulation from parking and exit the simulation to parking, and logic for public transit such as buses. These additions are implemented in a way that no further exchange of boundary information is necessary.

The implementation uses the so-called master-slave approach. Master-slave approach means that the simulation is started up by a master, which spawns slaves, distributes the workload to them, and keeps control of the general scheduling. Master-slave approaches often do not scale well with increasing numbers of CPUs since the workload of the master remains the same or even increases with increasing numbers of CPUs. For that reason, in Transims1999 the master has nearly no tasks except initialization and synchronization. Even the output to file is done in a decentralized fashion. With the numbers of CPUs that we have tested in practice, we have never observed the master being the bottleneck of the parallelization.

The actual implementation was done by defining descendent C++ classes of the C++ base classes provided in a Parallel Toolbox. The underlying communication library has interfaces for both PVM (Parallel Virtual Machine (PVM www page, accessed 2003)) and MPI (Message Passing Interface (MPI www page, accessed 2003)). The toolbox implementation is not specific to transportation simulations and thus beyond the scope of this paper. More information can be found in (Rickert, 1998).

25.3 Graph partitioning

Once we are able to handle split links, we need to partition the whole transportation network graph in an efficient way. Efficient means several competing things: Minimize the number of split links; minimize the number of other domains each CPU shares links with; equilibrate the computational load as much as possible.

One approach to domain decomposition is orthogonal recursive bi-section. Although less efficient than METIS (explained below), orthogonal bi-section is useful for explaining the general approach. In our case, since we cut in the middle of links, the first step is to accumulate computational loads at the nodes: each node gets a weight corresponding

[1] Instead of "split links", the terms "boundary links", "shared links", or "distributed links" are sometimes used. As is well known, some people use "edge" instead of "link".

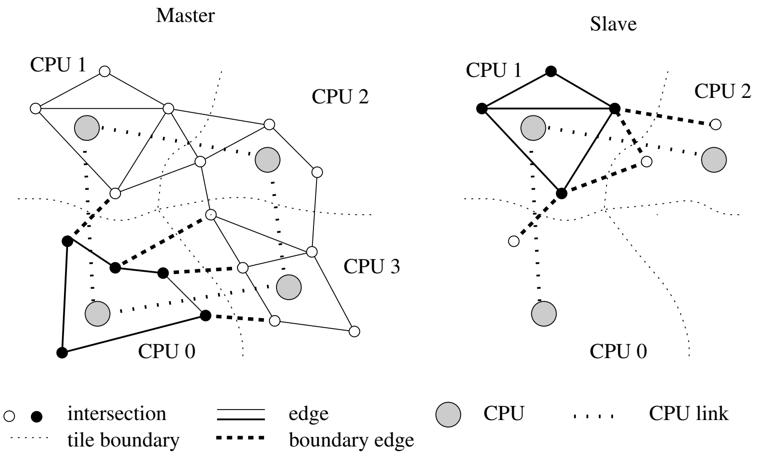


Figure 25.1: Domain decomposition of transportation network. *Left:* Global view. *Right:* View of a slave CPU. The slave CPU is only aware of the part of the network which is attached to its local nodes. This includes links which are shared with neighbor domains.

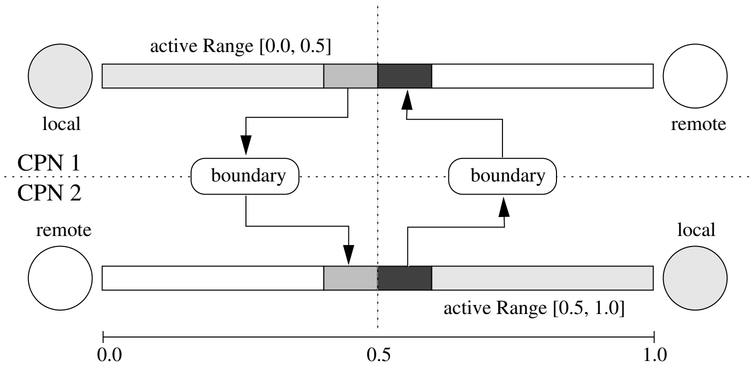


Figure 25.2: Distributed link.

to the computational load of all of its attached half-links. Nodes are located at their geographical coordinates. Then, a vertical straight line is searched so that, as much as possible, half of the computational load is on its right and the other half on its left. Then the larger of the two pieces is picked and cut again, this time by a horizontal line. This is recursively done until as many domains are obtained as there are CPUs available, see Fig. 25.4. It is immediately clear that under normal circumstances this will be most efficient for a number of CPUs that is a power of two. With orthogonal bi-section, we obtain compact and localized domains, and the number of neighbor domains is limited.

Another option is to use the METIS library for graph partitioning (see (www-users.cs.umn.edu/˜karypis/metis/, accessed 2003) and references therein). METIS uses multilevel partitioning. What that means is that first the graph is coarsened, then the coarsened graph is partitioned, and then it is uncoarsened again, while using an exchange heuristic at every uncoarsening step. The coarsening can for example be done via random matching, which means that first edges are randomly selected so that no two selected links share the same vertex, and then the two nodes at the end of each edge are collapsed into one. Once the graph is sufficiently collapsed, it is easy to find a good or optimal partitioning for the collapsed

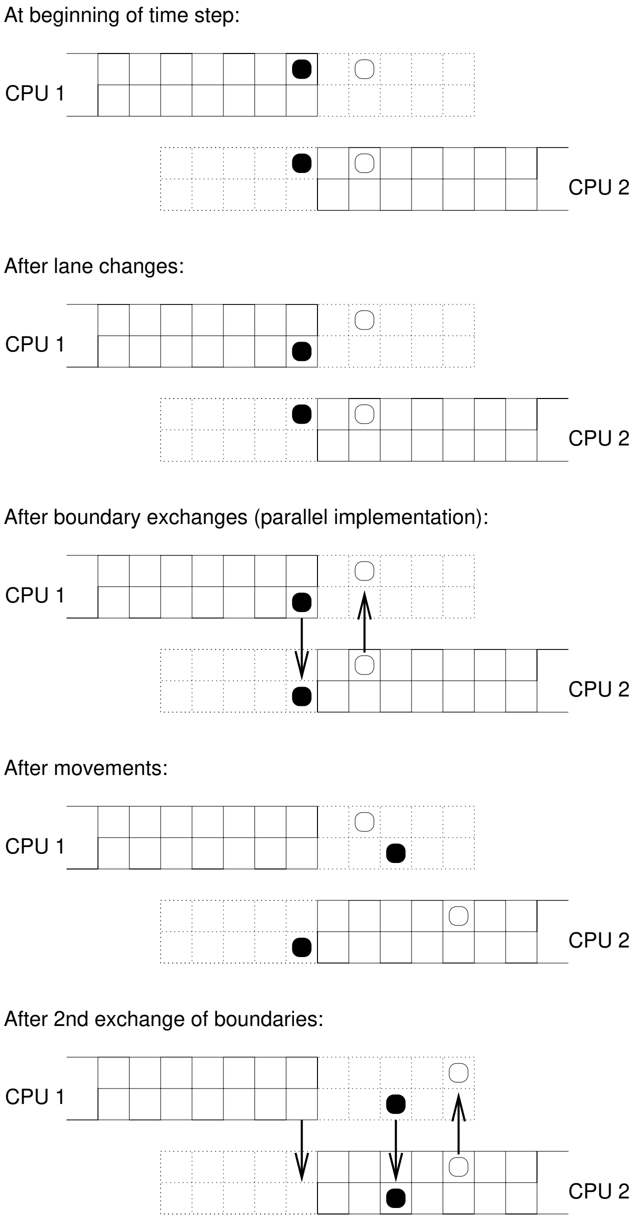


Figure 25.3: Example of parallel logic of a split link with two lanes. The figure shows the general logic of one time step. Remember that with a split link, one CPU is responsible for one half of the link and another CPU is responsible for the other half. These two halves are shown separately but correctly lined up. The dotted part is the "boundary region", which is where the link stores information from the other CPU. The arrows denote when information is transferred from one CPU to the other via boundary exchange.

graph. During uncoarsening, it is systematically tried if exchanges of nodes at the boundaries lead to improvements. "Standard" METIS uses multilevel recursive bisection: The initial graph is partitioned into two pieces, each of the two pieces is partitioned into two pieces each again, etc., until there are enough pieces. Each such split uses its own coarsening/uncoarsening sequence. k-METIS means that all k partitions are found during a single coarsening/uncoarsening sequence, which is considerably faster. It also produces more consistent and better results for large k.

METIS considerably reduces the number of split links, N_{spl}, as shown in Fig. 25.5. The figure shows the number of split links as a function of the number of domains for (i) orthogonal bi-section for a Portland network with 200 000 links, (ii) METIS decomposition for the same network, and (iii) METIS decomposition for a Portland network with 20 024 links. The network with 200 000 links is derived from the TIGER census data base, and will be used for the Portland case study for Transims. The network with 20 024 links is derived from the EMME/2 network that Portland is currently using. An example of the domains generated by METIS can be seen in Fig. 25.6; for example, the algorithm now picks up the fact that cutting along the rivers in Portland should be of advantage since this results in a small number of split links.

We also show data fits to the METIS curves, $N_{spl} = 250\,p^{0.59}$ for the 200 000 links network and $N_{spl} = 140\,p^{0.59} - 140$ for the 20 024 links network, where p is the number of domains. We are not aware of any theoretical argument for the shapes of these curves for METIS. It is however easy to see that, for orthogonal bisection, the scaling of N_{spl} has to be $\sim p^{0.5}$. Also, the limiting case where each node is on a different CPU needs to have the same N_{spl} both for bisection and for METIS. In consequence, it is plausible to use a scaling form of p^{α} with $\alpha > 0.5$. This is confirmed by the straight line for large p in the log-log-plot of Fig. 25.5. Since for $p = 1$, the number of split links N_{spl} should be zero, for the 20 024 links network we use the equation $A\,p^{\alpha} - A$, resulting in $N_{spl} = 140\,p^{0.59} - 140$. For the 200 000 links network, the resulting fit is so bad that we did not add the negative term. This leads to a kink for the corresponding curves in Fig. 25.12.

Such an investigation also allows to compute the theoretical efficiency based on the graph partitioning. Efficiency is optimal if each CPU gets exactly the same computational load. However, because of the granularity of the entities (nodes plus attached half-links) that we distribute, load imbalances are unavoidable, and they become larger with more CPUs. We define the resulting theoretical efficiency due to the graph partitioning as

$$e_{dmn} := \frac{\text{load on optimal partition}}{\text{load on largest partition}}\,, \tag{25.1}$$

where the load on the optimal partition is just the total load divided by the number of CPUs. We then calculated this number for actual partitionings of both of our 20 024 links and of our 200 000 links Portland networks, see Fig. 25.7. The result means that, according to this measure alone, our 20 024 links network would still run efficiently on 128 CPUs, and our 200 000 links network would run efficiently on up to 1024 CPUs.

25.4 Adaptive Load Balancing

In the last section, we explained how the street network is partitioned into domains that can be loaded onto different CPUs. In order to be efficient, the loads on different CPUs should be as similar as possible. These loads do however depend on the actual vehicle traffic in the respective domains. Since we are doing iterations, we are running similar traffic scenarios over and over again. We use this feature for an adaptive load balancing: During run time we collect the execution time of each link and each intersection (node). The statistics are output to file. For the next run of the micro-simulation, the file is fed back to the partitioning algorithm. In that iteration, instead of using the link lengths as load estimate, the actual execution times are used as distribution criterion. Fig. 25.8 shows the new domains after such a feedback (compare to Fig. 25.4).

To verify the impact of this approach we monitored the execution times per time-step throughout the simulation period. Figure 25.9 depicts the results of one of the iteration

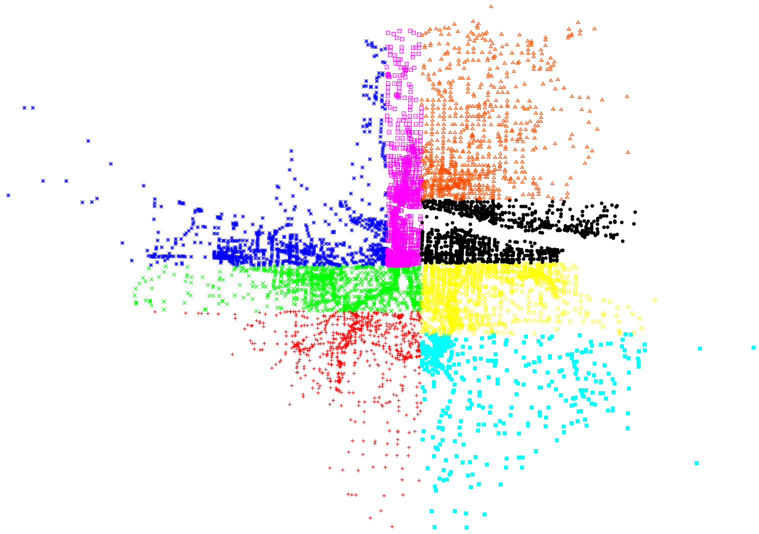


Figure 25.4: Orthogonal bi-section for Portland 20 024 links network.

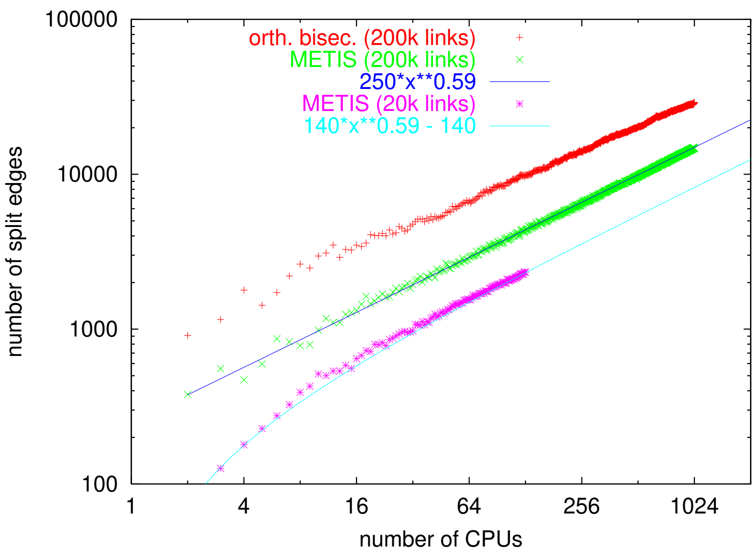


Figure 25.5: Number of split links as a function of the number of CPUs. The top curve shows the result of orthogonal bisection for the 200 000 links network. The middle curve shows the result of METIS for the same network – clearly, the use of METIS results in considerably fewer split links. The bottom curve shows the result for the Portland 20 024 links network when again using METIS. The theoretical scaling for orthogonal bisection is $N_{spl} \sim \sqrt{p}$, where p is the number of CPUs. Note that for $p \to N_{links}$, N_{spl} needs to be the same for both graph partitioning methods.

series. For iteration 1, the load balancer uses the link lengths as criterion. The execution times are low until congestion appears around 7:30 am. Then, the execution times increase fivefold from 0.04 sec to 0.2 sec. In iteration 2 the execution times are almost independent of the simulation time. Note that due to the equilibration, the execution times

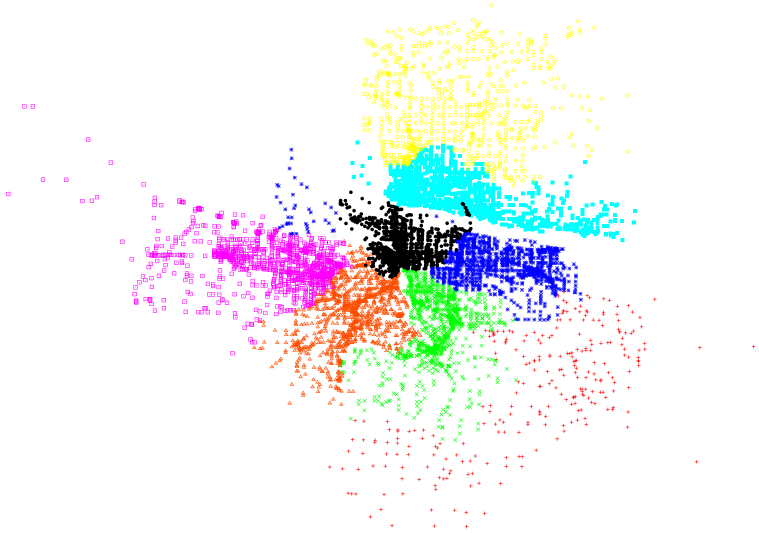


Figure 25.6: Partitioning by METIS. Compare to Fig. 25.4.

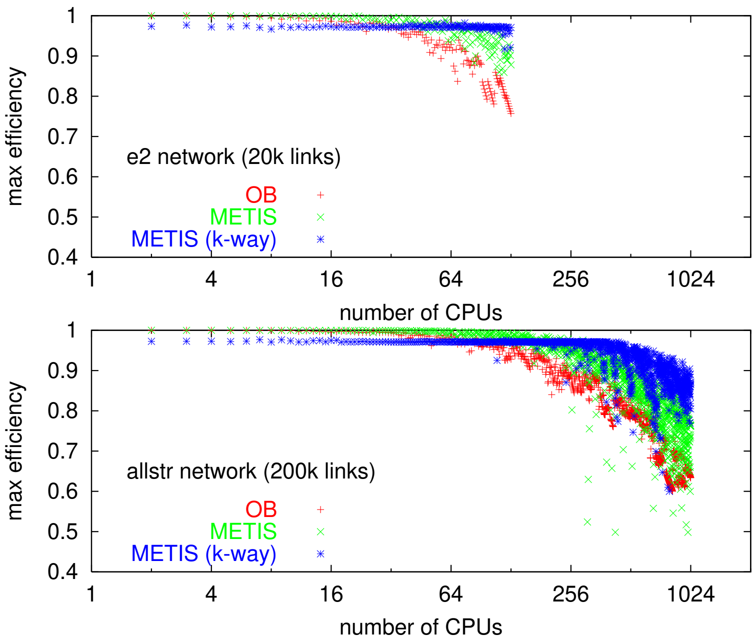


Figure 25.7: *Top:* Theoretical efficiency for Portland network with 20 024 links. *Bottom:* Theoretical efficiency for Portland network with 200 000 links. "OB" refers to orthogonal bisection. "METIS (k-way)" refers to an option in the METIS library.

for early simulation hours increase from 0.04 sec to 0.06 sec, but this effect is more than compensated later on.

The figure also contains plots for later iterations (11, 15, 20, and 40). The improvement of execution times is mainly due to the route adaptation process: congestion is reduced and the average vehicle density is lower. On the machine sizes where we have tried it (up to 16 CPUs), adaptive load balancing led to performance improvements up to a factor of

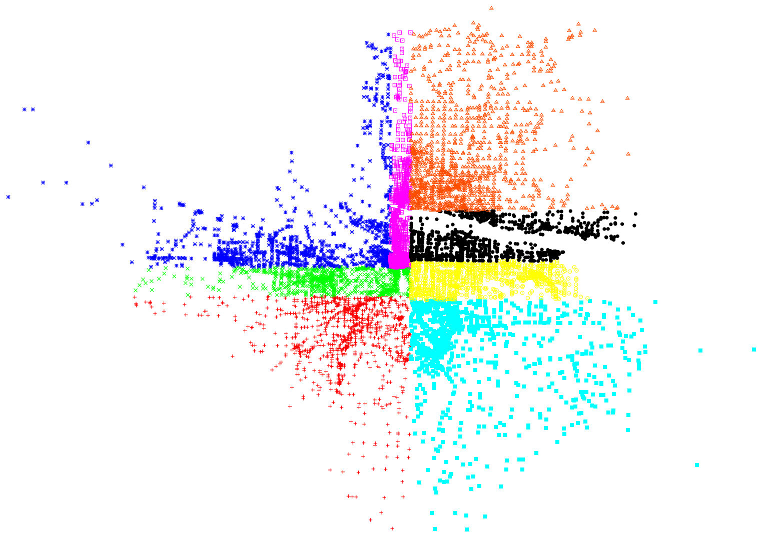


Figure 25.8: Partitioning after adaptive load balancing. Compare to Fig. 25.4.

Figure 25.9: Execution times with external load feedback. These results were obtained during the Dallas case study (Beckman et al, 1997; Rickert, 1998).

1.8. It should become more important for larger numbers of CPUs since load imbalances have a stronger effect there.

25.5 Performance prediction for the Transims micro-simulation

It is possible to systematically predict the performance of parallel micro-simulations (e.g. (Jakobs and Gerling, 1993; Nagel and Schleicher, 1994)). For this, several assumptions about the computer architecture need to be made. In the following, we demonstrate the derivation of such predictive equations for coupled workstations and for parallel super-computers.

The method for this is to systematically calculate the wall clock time for one time step of the micro-simulation. We start by assuming that the time for one time step has contributions from computation, T_{cmp}, and from communication, T_{cmm}. If these do not overlap, as is reasonable to assume for coupled workstations, we have

$$T(p) = T_{cmp}(p) + T_{cmm}(p) \ , \tag{25.2}$$

where p is the number of CPUs.[2]

Time for computation is assumed to follow

$$T_{cmp}(p) = \frac{T_1}{p} \cdot \Big(1 + f_{ovr}(p) + f_{dmn}(p)\Big) \ . \tag{25.3}$$

[2]For simplicity, we do not differentiate between CPUs and computational nodes. Computational nodes can have more than one CPU — an example is a network of coupled PCs where each PC has Dual CPUs.

Here, T_1 is the time of the same code on one CPU (assuming a problem size that fits on available computer memory); p is the number of CPUs; f_{ovr} includes overhead effects (for example, split links need to be administered by *both* CPUs); $f_{dmn} = 1/e_{dmn} - 1$ includes the effect of unequal domain sizes discussed in Sec. 25.3.

Time for communication typically has two contributions: Latency and bandwidth. Latency is the time necessary to initiate the communication, and in consequence it is independent of the message size. Bandwidth describes the number of bytes that can be communicated per second. So the time for one message is

$$T_{msg} = T_{lt} + \frac{S_{msg}}{b} \ , \tag{25.4}$$

where T_{lt} is the latency, S_{msg}, is the message size, and b is the bandwidth.

However, for many of today's computer architectures, bandwidth is given by at least two contributions: node bandwidth, and network bandwidth. Node bandwidth is the bandwidth of the connection from the CPU to the network. If two computers communicate with each other, this is the maximum bandwidth they can reach. For that reason, this is sometimes also called the "point-to-point" bandwidth.

The network bandwidth is given by the technology and topology of the network. Typical technologies are 10 Mbit Ethernet, 100 Mbit Ethernet, FDDI, etc. Typical topologies are bus topologies, switched topologies, two-dimensional topologies (e.g. grid/torus), hypercube topologies, etc. A traditional Local Area Network (LAN) uses 10 Mbit Ethernet, and it has a shared bus topology. In a shared bus topology, all communication goes over the same medium; that is, if several pairs of computers communicate with each other, they have to share the bandwidth.

For example, in our 100 Mbit FDDI network (i.e. a network bandwidth of $b_{net} = 100$ Mbit) at Los Alamos National Laboratory, we found node bandwidths of about $b_{nd} = 40$ Mbit. That means that two pairs of computers could communicate at full node bandwidth, i.e. using 80 of the 100 Mbit/sec, while three or more pairs were limited by the network bandwidth. For example, five pairs of computers could maximally get $100/5 = 20$ Mbit/sec each.

A switched topology is similar to a bus topology, except that the network bandwidth is given by the backplane of the switch. Often, the backplane bandwidth is high enough to have all nodes communicate with each other at full node bandwidth, and for practical purposes one can thus neglect the network bandwidth effect for switched networks.

If computers become massively parallel, switches with enough backplane bandwidth become too expensive. As a compromise, such supercomputers usually use a communications topology where communication to "nearby" nodes can be done at full node bandwidth, whereas global communication suffers some performance degradation. Since we partition our traffic simulations in a way that communication is local, we can assume that we do communication with full node bandwidth on a supercomputer. That is, on a parallel supercomputer, we can neglect the contribution coming from the b_{net}-term. This assumes, however, that the allocation of street network partitions to computational nodes is done in some intelligent way which maintains locality.

As a result of this discussion, we assume that the communication time per time step is

$$T_{cmm}(p) = N_{sub} \cdot \left(n_{nb}(p)\, T_{lt} + \frac{N_{spl}(p)}{p} \frac{S_{bnd}}{b_{nd}} + N_{spl}(p)\, \frac{S_{bnd}}{b_{net}} \right) , \tag{25.5}$$

which will be explained in the following paragraphs. N_{sub} is the number of sub-time-steps. As discussed in Sec. 25.2, we do two boundary exchanges per time step, thus $N_{sub} = 2$ for the 1999 Transims micro-simulation implementation.

n_{nb} is the number of neighbor domains each CPU talks to. All information which goes to the same CPU is collected and sent as a single message, thus incurring the latency only once per neighbor domain. For $p = 1$, n_{nb} is zero since there is no other domain to communicate with. For $p = 2$, it is one. For $p \to \infty$ and assuming that domains are always connected, Euler's theorem for planar graphs says that the average number of neighbors cannot become more than six. Based on a simple geometric argument, we use

$$n_{nb}(p) = 2\,(3\sqrt{p} - 1)\,(\sqrt{p} - 1)/p\ , \tag{25.6}$$

which correctly has $n_{nb}(1) = 0$ and $n_{nb} \to 6$ for $p \to \infty$. Note that the METIS library for graph partitioning (Sec. 25.3) does not necessarily generate connected partitions, making this potentially more complicated.

T_{lt} is the latency (or start-up time) of each message. T_{lt} between 0.5 and 2 milliseconds are typical values for PVM on a LAN (Rickert, 1998; Dongarra et al., 1998).

Next are the terms that describe our two bandwidth effects. $N_{spl}(p)$ is the number of split links in the whole simulation; this was already discussed in Sec. 25.3 (see Fig. 25.5). Accordingly, $N_{spl}(p)/p$ is the number of split links per computational node. S_{bnd} is the size of the message per split link. b_{nd} and b_{net} are the node and network bandwidths, as discussed above.

In consequence, the combined time for one time step is

$$T(p) = \frac{T_1}{p}\Big(1 + f_{ovr}(p) + f_{dmn}(p)\Big) + \tag{25.7}$$

$$N_{sub} \cdot \left(n_{nb}(p)\,T_{lt} + \frac{N_{spl}(p)}{p}\,\frac{S_{bnd}}{b_{nd}} + N_{spl}(p)\,\frac{S_{bnd}}{b_{net}} \right)\ . \tag{25.8}$$

According to what we have discussed above, for $p \to \infty$ the number of neighbors scales as $n_{nb} \sim const$ and the number of split links in the simulation scales as $N_{spl} \sim \sqrt{p}$. In consequence for f_{ovr} and f_{dmn} small enough, we have:

- for a shared or bus topology, b_{net} is relatively small and constant, and thus
$$T(p) \sim \frac{1}{p} + 1 + \frac{1}{\sqrt{p}} + \sqrt{p} \to \sqrt{p}\ ; \tag{25.9}$$
- for a switched or a parallel supercomputer topology, we assume $b_{net} = \infty$ and obtain
$$T(p) \sim \frac{1}{p} + 1 + \frac{1}{\sqrt{p}} \to 1\ . \tag{25.10}$$

Thus, in a shared topology, adding CPUs will eventually *in*crease the simulation time, thus making the simulation *slower*. In a non-shared topology, adding CPUs will eventually not make the simulation any faster, but at least it will not be detrimental to computational speed. The dominant term in a shared topology for $p \to \infty$ is the network bandwidth; the dominant term in a non-shared topology is the latency.

The curves in Fig. 25.10 are results from this prediction for a switched 100 Mbit Ethernet LAN; dots and crosses show actual performance results. The top graph shows the time for one time step, i.e. $T(p)$, and the individual contributions to this value. The bottom graph shows the real time ratio (RTR)

$$rtr(p) := \frac{\Delta t}{T(p)} = \frac{1\ sec}{T(p)}\ , \tag{25.11}$$

which says how much faster than reality the simulation is running. Δt is the duration a simulation time step, which is $1\ sec$ in Transims1999. The values of the free parameters are:

- **Hardware-dependent parameters**. We assume that the switch has enough bandwidth so that the effect of b_{net} is negligeable. Other hardware parameters are $T_{lt} = 0.8$ ms and $b_{nd} = 50$ Mbit/s.[3]

- **Implementation-dependent parameters**. The number of message exchanges per time step is $N_{sub} = 2$.

- **Scenario-dependent parameters**. Except when noted, our performance predictions and measurements refer to the Portland 20 024 links network. We use, for the number of split links, $N_{spl}(p) = 140 \cdot p^{0.59} - 140$, as explained in Sec. 25.3.

- **Other Parameters.** The message size depends on the plans format (which depends on the software design and implementation), on the typical number of links in a plan, and on the frequency per link of vehicles migrating from one CPU to another. We use $S_{bnd} = 200\ Bytes$. This is an average number; it includes all the information that needs to be sent when a vehicle migrates from one CPU to another. The new Transims multi-modal plans format easily has 200 entries per driver and trip, resulting in 800 bytes of information just for the plan. In addition, there is information about the vehicle (ID, speed, maximum acceleration, etc.); however, not in every time step a vehicle is migrated across a boundary on every split link. In principle it is however possible to compress the plans information, so improvements are possible here in the future. Also, we have not explicitely modelled simulation output, which is indeed a performance issue on Beowulf clusters.

These parameters were obtained in the following way: First, we obtained plausible values via systematic communication tests using messages similar to the ones used in the actual simulation (Rickert, 1998). Then, we ran the simulation without any vehicles (see below) and adapted our values accordingly. Running the simulation without vehicles means that we have a much better control of S_{bnd}. In practice, the main result of this step was to set t_{lat} to 0.8 msec, which is plausible when compared to the hardware value of 0.5 msec. Last, we ran the simulations with vehicles and adjusted S_{bnd} to fit the data. — In consequence, for the switched 100 Mbit Ethernet configurations, within the data range our curves are model fits to the data. Outside the data range and for other configurations, the curves are model-based predictions.

The plot (Fig. 25.10) shows that even something as relatively profane as a combination of regular Pentium CPUs using a switched 100Mbit Ethernet technology is quite capable in reaching good computational speeds. For example, with 16 CPUs the simulation runs 40 times faster than real time; the simulation of a 24 hour time period would thus take 0.6 hours. These numbers refer, as said above, to the Portland 20 024 links network. Included in the plot (black dots) are measurements with a compute cluster that corresponds to this architecture. The triangles with lower performance for the same number of CPUs come from using dual instead of single CPUs on the computational nodes. Note that the curve levels out at about forty times faster than real time, no matter what the number of CPUs. As one can see in the top figure, the reason is the latency term, which eventually consumes nearly all the time for a time step. This is one of the important elements where parallel supercomputers are different: For example the Cray T3D has a more than a factor of ten lower latency under PVM (Dongarra et al., 1998).

As mentioned above, we also ran the same simulation without any vehicles. In the Transims1999 implementation, the simulation sends the contents of each CA boundary region to the neighboring CPU even when the boundary region is empty. Without compression, this is five integers for five sites, times the number of lanes, resulting in about 40 bytes per split edge, which is considerably less than the 800 bytes from above. The

[3] Our measurements have consistently shown that node bandwidths are lower than network bandwidths. Even CISCO itself specifies 148 000 packets/sec, which translates to about 75 Mbit/sec, for the 100 Mbit switch that we use.

results are shown in Fig. 25.11. Shown are the computing times with 1 to 15 single-CPU slaves, and the corresponding real time ratio. Clearly, we reach better speed-up without vehicles than with vehicles (compare to Fig. 25.10). Interestingly, this does not matter for the maximum computational speed that can be reached with this architecture: Both with and without vehicles, the maximum real time ratio is about 80; it is simply reached with a higher number of CPUs for the simulation with vehicles. The reason is that eventually the only limiting factor is the network latency term, which does not have anything to do with the *amount* of information that is communicated.

Fig. 25.12 (top) shows some predicted real time ratios for other computing architectures. For simplicity, we assume that all of them except for one special case explained below use the same 500 MHz Pentium compute nodes. The difference is in the networks: We assume 10 Mbit non-switched, 10 Mbit switched, 1 Gbit non-switched, and 1 Gbit switched. The curves for 100 Mbit are in between and were left out for clarity; values for switched 100 Mbit Ethernet were already in Fig. 25.10. One clearly sees that for this problem and with today's computers, it is nearly impossible to reach *any* speed-up on a 10 Mbit Ethernet, even when switched. Gbit Ethernet is somewhat more efficient than 100 Mbit Ethernet for small numbers of CPUs, but for larger numbers of CPUs, switched Gbit Ethernet saturates at exactly the same computational speed as the switched 100 Mbit Ethernet. This is due to the fact that we assume that latency remains the same – after all, there was no improvement in latency when moving from 10 to 100 Mbit Ethernet. FDDI is supposedly even worse (Dongarra et al., 1998).

The thick line in Fig. 25.12 corresponds to the ASCI Blue Mountain parallel supercomputer at Los Alamos National Laboratory. On a per-CPU basis, this machine is slower than a 500 MHz Pentium. The higher bandwidth and in particular the lower latency make it possible to use higher numbers of CPUs efficiently, and in fact one should be able to reach a real time ratio of 128 according to this plot. By then, however, the granularity effect of the unequal domains (Eq. (25.1), Fig. 25.7) would have set in, limiting the computational speed probably to about 100 times real time with 128 CPUs. We actually have some speed measurements on that machine for up to 96 CPUs, but with a considerably slower code from summer 1998. We omit those values from the plot in order to avoid confusion.

Fig. 25.12 (bottom) shows predictions for the higher fidelity Portland 200 000 links network with the same computer architectures. The assumption was that the time for one time step, i.e. T_1 of Eq. (25.3), increases by a factor of eight due to the increased load. This has not been verified yet. However, the general message does not depend on the particular details: When problems become larger, then larger numbers of CPUs become more efficient. Note that we again saturate, with the switched Ethernet architecture, at 80 times faster than real time, but this time we need about 64 CPUs with switched Gbit Ethernet in order to get 40 times faster than real time — for the smaller Portland 20 024 links network with switched Gbit Ethernet we would need 8 of the same CPUs to reach the same real time ratio. In short and somewhat simplified: As long as we have enough CPUs, we can micro-simulate road networks of *arbitrarily largesize*, with hundreds of thousands of links and more, 40 times faster than real time, even without supercomputer hardware. — Based on our experience, we are confident that these predictions will be lower bounds on performance: In the past, we have always found ways to make the code more efficient.

25.6 Speed-up and efficiency

We have cast our results in terms of the real time ratio, since this is the most important quantity when one wants to get a practical study done. In this section, we will trans-

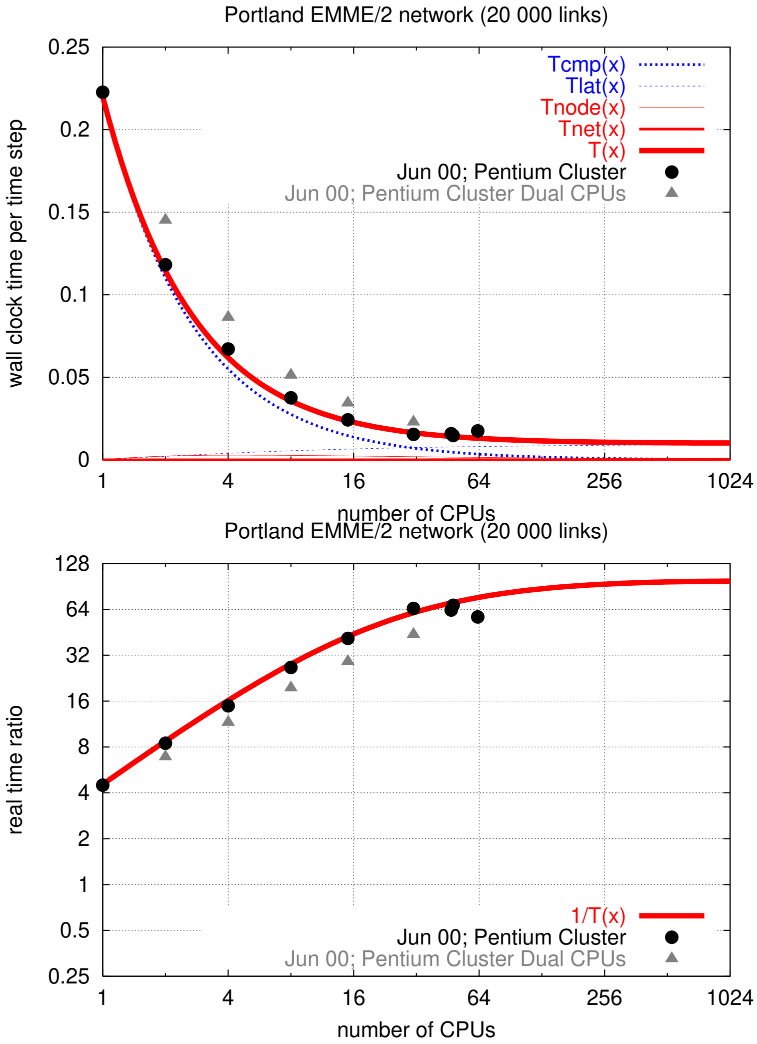


Figure 25.10: 100 Mbit switched Ethernet LAN. *Top:* Individual time contributions. *Bottom:* Corresponding Real Time Ratios. The black dots refer to actually measured performance when using one CPU per cluster node; the crosses refer to actually measured performance when using dual CPUs per node (the y-axis still denotes the number of CPUs used). The thick curve is the prediction according to the model. The thin lines show the individual time contributions to the thick curve.

late our results into numbers of speed-up, efficiency, and scale-up, which allow easier comparison for computing people.

Let us define speed-up as

$$S(p) := \frac{T(1)}{T(p)} , \tag{25.12}$$

where p is again the number of CPUs, $T(1)$ is the time for one time-step on one CPU, and $T(p)$ is the time for one time step on p CPUs. Depending on the viewpoint, for $T(1)$ one uses either the running time of the parallel algorithm on a single CPU, or the fastest existing sequential algorithm. Since Transims has been designed for parallel computing and since there is no sequential simulation with exactly the same properties, $T(1)$ will be

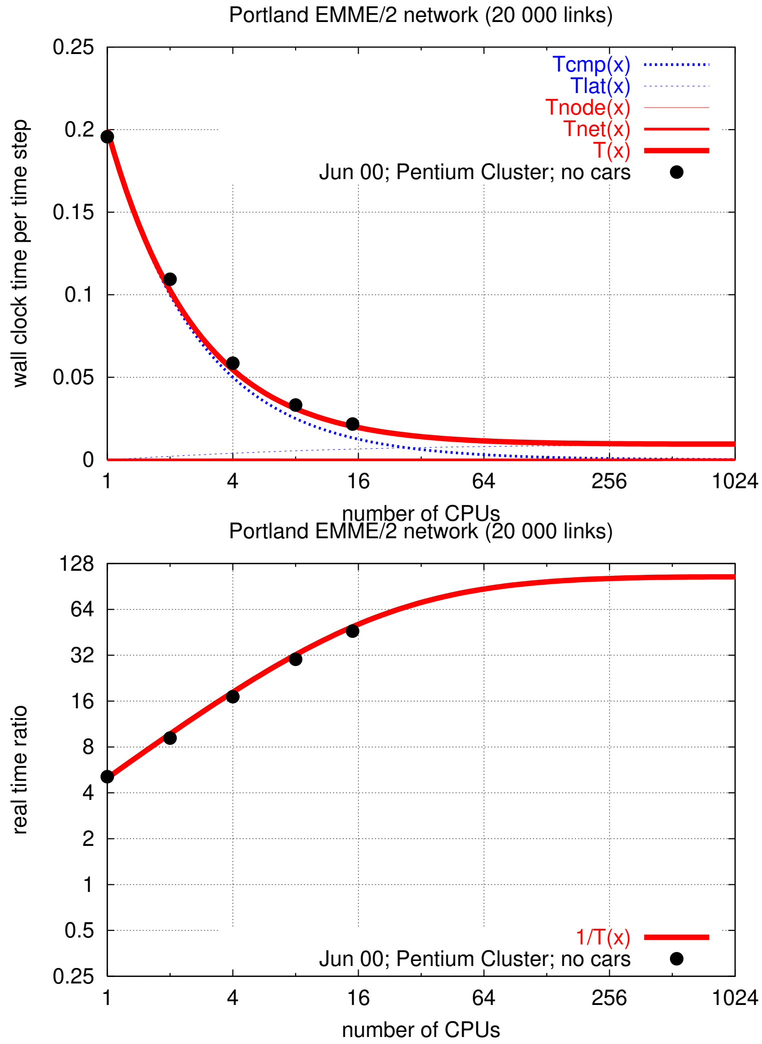


Figure 25.11: 100 Mbit switched Ethernet LAN; simulation without vehicles. *Top:* Individual time contributions. *Bottom:* Corresponding Real Time Ratios. The same remarks as to Fig. 25.10 apply. In particular, black dots show measured performance, whereas curves show predicted performance.

the running time of the parallel algorithm on a single CPU. For time-stepped simulations such as used here, the difference is expected to be small.[4]

Now note again that the real time ratio is $rtr(p) = 1\ sec/T(p)$. Thus, in order to obtain the speed-up from the real time ratio, one has to multiply all real time ratios by $T(1)/(1\ sec)$. On a logarithmic scale, a multiplication corresponds to a linear shift. In consequence, speed-up curves can be obtained from our real time ratio curves by shifting the curves up or down so that they start at one.

This also makes it easy to judge if our speed-up is linear or not. For example in Fig. 25.12 bottom, the curve which starts at 0.5 for 1 CPU should have an RTR of 2 at 4 CPU, an RTR of 8 at 16 CPUs, etc. Downward deviations from this mean sub-linear speed-

[4]An event-driven simulation could be a counter-example: Depending on the implementation, it could be extremely fast on a single CPU up to medium problem sizes, but slow on a parallel machine.

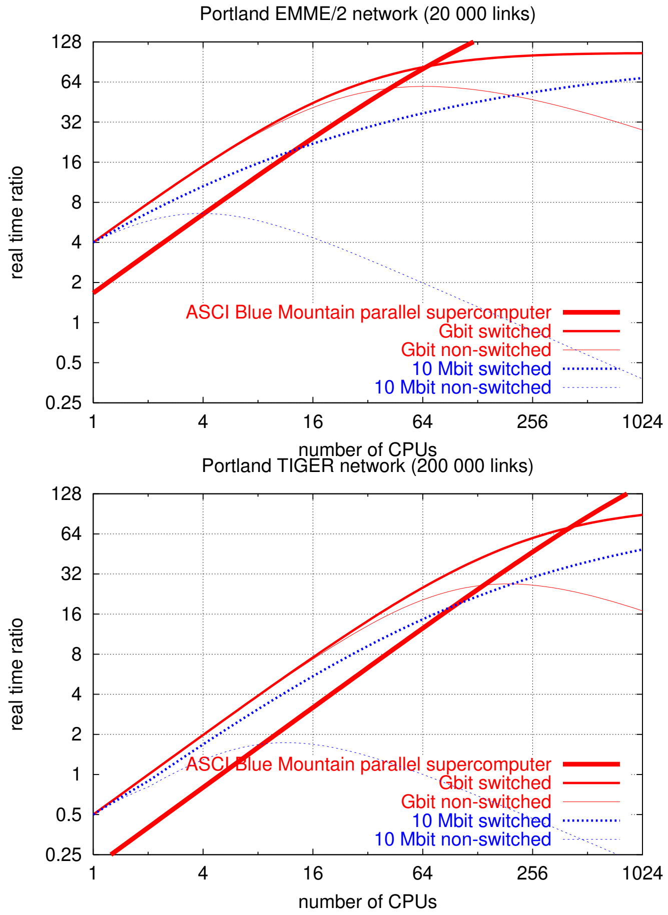


Figure 25.12: Predictions of real time ratio for other computer configurations. *Top:* With Portland EMME/2 network (20 024 links). *Bottom:* With Portland TIGER network (200 000 links). Note that for the switched configurations and for the supercomputer, the saturating real time ratio is the same for both network sizes, but it is reached with different numbers of CPUs. This behavior is typical for parallel computers: They are particularly good at running larger and larger problems within the same computing time. — All curves in both graphs are predictions from our model. We have some performance measurements for the ASCI maschine, but since they were done with an older and slower version of the code, they are omitted in order to avoid confusion.

up. Such deviations are commonly described by another number, called efficiency, and defined as

$$E(p) := \frac{T(1)/p}{T(p)} . \qquad (25.13)$$

Fig. 25.13 contains an example. Note that this number contains no new information; it is just a re-interpretation. Also note that in our logarithmic plots, $E(p)$ will just be the difference to the diagonal $p\,T(1)$. Efficiency can point out where improvements would be useful.

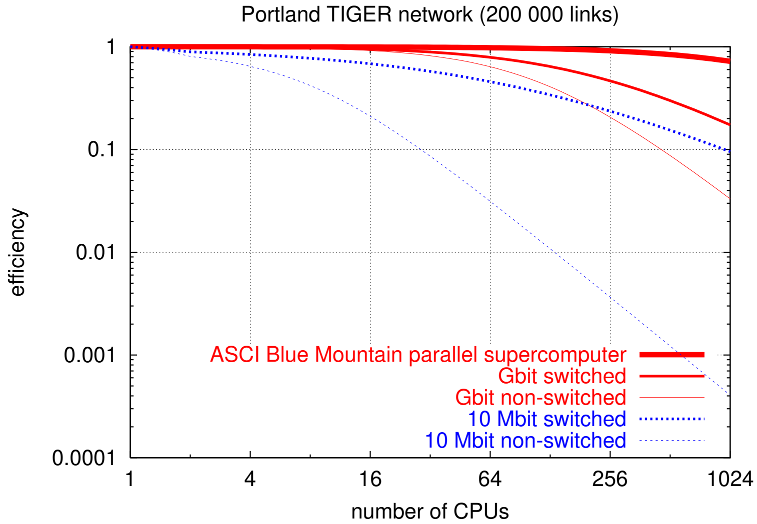


Figure 25.13: Efficiency for the same configurations as in Fig. 25.12 bottom. Note that the curves contain exactly the same information.

25.7 Other modules

As explained in the introduction, a micro-simulation in a software suite for transportation planning would have to be run many times ("feedback iterations") in order to achieve consistency between modules. For the microsimulation alone, and assuming our 16 CPU-machine with switched 100 Mbit Ethernet, we would need about 30 hours of computing time in order to simulate 24 hours of traffic fifty times in a row. In addition, we have the contributions from the other modules (routing, activities generation). In the past, these have never been a larger problem than the micro-simulation, for several reasons:

- The algorithms of the other modules by themselves did significantly less computation than the micro-simulation.
- Even when these algorithms start using considerable amounts of computer time, they are "trivially" parallelizable by simply distributing the households across CPUs.[5]
- In addition, during the iterations we never replan more than about 10% of the population, saving additional computer time.

In summary, the Transims modules besides the traffic micro-simulation currently do not contribute significantly to the computational burden; in consequence, the computational performance of the traffic micro-simulation is a good indicator of the overall performance of the simulation system.

25.8 Summary

This paper explains the parallel implementation of the Transims micro-simulation. Since other modules are computationally less demanding and also simpler to parallelize, the

[5] This is possible because of the specific purpose Transims is designed for. In real time applications, where absolute speed between request and response matters, the situation is different (Chabini, 1998a).

parallel implementation of the micro-simulation is the most important and most complicated piece of parallelization work. The parallelization method for the Transims micro-simulation is domain decomposition, that is, the network graph is cut into as many domains as there are CPUs, and each CPU simulates the traffic on its domain. We cut the network graph in the middle of the links rather than at nodes (intersections), in order to separate the traffic dynamics complexity at intersections from the complexity of the parallel implementation. We explain how the cellular automata (CA) or any technique with a similar time depencency scheduling helps to design such split links, and how the message exchange in Transims works.

The network graph needs to be partitioned into domains in a way that the time for message exchange is minimized. Transims uses the METIS library for this goal. Based on partitionings of two different networks of Portland (Oregon), we calculate the number of CPUs where this approach would become inefficient just due to this criterion. For a network with 200 000 links, we find that due to this criterion alone, up to 1024 CPUs would be efficient. We also explain how the Transims micro-simulation adapts the partitions from one run to the next during feedback iterations (adaptive load balancing).

We finally demonstrate how computing time for the Transims micro-simulation (and therefore for all of Transims) can be systematically predicted. An important result is that the Portland 20 024 links network runs about 40 times faster than real time on 16 dual 500 MHz Pentium computers connected via switched 100 Mbit Ethernet. These are regular desktop/LAN technologies. When using the next generation of communications technology, i.e. Gbit Ethernet, we predict the same computing speed for a much larger network of 200 000 links with 64 CPUs.

[[in particular look at Nurhan's "mistakes" and clarify this stuff. In particular, clarify where on a beowulf the time is spent, and how this degrades performance if one uses a master etc.]]

[[maybe also explain quantify.]]

[[multi-crit metis??]]

[[threads?? at least mention and discuss.]]

Chapter 26

Distributed computing and truly distributed intelligence

Once the traffic micro-simulation is parallelized, it becomes considerably more difficult to add within-day replanning. As long as one runs everything on a single CPU, it is in principle possible to write one monolithic software package. In such a software, an agent who wants to change plans calls a subroutine to compute a new plan, and during this time the computation of the traffic dynamics is suspended. On a parallel computer, if one traveler on one CPU does this, *all other* CPUs have to suspend the traffic simulation since it is not possible (or very difficult) to have simulated time continue asynchronously (Fig. 26.1 left).

A better approach is to have the re-planning module on a different CPU. The traveler then sends out the re-planning request to that CPU, and the traffic simulation keeps going (Figs. ?? and 26.1 right). Eventually, the re-planning will be finished, and its result will be sent to the simulated traveler, who picks it up and starts acting on it. An experimental implementation of this using UDP (User Datagram Protocol) for communication shows that it is possible to transmit up to 100 000 requests per second per CPU (Gloor, 2001), which is far above any number that is relevant for practical applications. This demonstrates that such a design is feasible and efficient.

Race conditions

Some readers may have noticed that success of the re-planning operation is not guaranteed. For example, the new plan may say to make a turn at a specific intersection, and by the time the new plan reaches the traveler, she/he may have driven past that point. Such situations are however not unusual in real life – how often does one recognize that a different decision some time ago would have been beneficial. Thus, in our view the key to success for large scale applications it to not fight asynchronous effects but to use them to advantage. For example, once it is accepted that such messages can arrive late, it is also not a problem to not have them arrive at all, which greatly simplifies message passing.

No memory problems etc.

An additional advantage of such a distributed design is that the implementation of a separate "mental map" (Sec. 31.3) for each individual traveler does not run into memory or CPU-time problems. Specific route guidance services can be simulated in a similar way. Also, non-local interaction between travelers becomes a matter of direct interaction between the corresponding "strategic" CPUs, without involving the rest of the computational engine. This occurs for example for ride sharing, or when family members re-organize the kindergarten pick-up when plans have changed during the day, and will

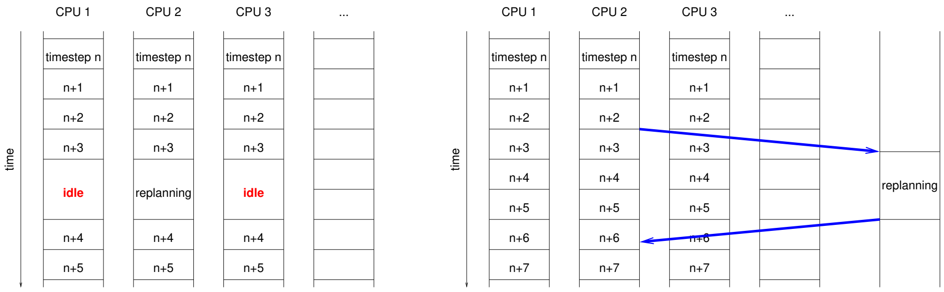


Figure 26.1: Parallel implementation of within-day replanning. LEFT: Implementation as subroutine of parallel traffic simulation. RIGHT: Implementation via separate plans server.

necessitate complicated negotiations between agents. However, neither the models nor the computational methods for this are developed.

Similarity to robot design and humans

This design is similar to many robot designs, where the robots are autonomous on short time scales (tactical level) while they are connected via wireless communication to a more powerful computer for more difficult and more long-term time scales (strategic level); see, e.g., Ref. (Kim, 1997) for robot soccer. Also, the human body is organized along these lines – for example, in ball catching, it seems that the brain does an approximate pre-“computation” of the movements of the hands, while the hands themselves (and autonomously) perform the fine-tuning of the movements as soon as the ball touches them and haptic information is available (Sternad). This approach is necessitated by the relatively slow message passing time between brain and hands, which is of the order of 1/10 sec, which is much too slow to directly react to haptic information (Rothwell, 1994).

That is, in summary we have a design where there is some kind of “real world dynamics” (the traffic simulation), which keeps going at its own pace. Agents can make strategic decisions, which may take time, but the world around them will keep going, meaning that they will have to continue driving, or deliberately park the car. As pointed out, such an architecture is very well supported by current distributed computers, although the actual implementation still needs to be done.

Part IV

Some background

Chapter 27

Traffic flow theory

[[In fact, I need something like simple micro-models, than simple fdyn, then more micro-models, etc. ???]]

27.1 Introduction

This text has started with a minimal representation of traffic on a link, the single-lane deterministic CA with maximum speed one. We have then explored ways to make that model more realistic, for example with respect to fundamental diagrams, or with respect to multi-lane traffic. The focus of this chapter will be to provide some basic underlying theory. Understanding some theory is necessary in particular if one wants to use simple models, because then one needs to understand their deficiencies and the consequences of this.

27.2 Traffic flow measurements

It was already pointed out in Sec. 17.3 that important real world quantities for traffic are flow and density. A third quantity is speed. In fact, there are two different ways to measure traffic: space-averaged measurements, and point ($=$ spot) measurements. The space-averaged measurements are done at specific points in time, and they correspond to what one is used to from, say, fluid-dynamics. The point measurements are closer to what is measured in reality: A sensor, e.g. an induction loop, usually covers only a small amount of space. It is common use to average point measurements over sometime T, for example $T = 60\ sec$ or $T = 5\ min$.[1] These differences are not particularly intereresting, but they are necessary to avoid some caveats.

27.2.1 Speed

The two measurements are:

- **Space-mean speed**, also called **travel velocity**:

$$v_L = \frac{1}{N_L} \sum_{i=1}^{N_L} v_i \ . \qquad (27.1)$$

[1] From a theoretical perspective, it is questionable if this averaging is a good idea. It is however necessary to compare with field data.

Thus, one averages over a stretch of road of length L.

- Point velocity, also called **spot speed** or **instantaneous velocity**. We observe at a fixed position, and we average over the velocities of all vehicles that pass by. When N_T is the number of vehicles that passed by, then spot speed is

$$\tilde{v}_T = \frac{1}{N_T} \sum v_i \; . \tag{27.2}$$

One can immediately see that there is a difference between space-mean speed and spot speed by noting that space-mean speed includes vehicles of speed zero into the average while spot speed does not. If, however, all vehicles always have the same velocity, then both measurements lead to the same result. The formal relationship is a bit more complicated.[2]

Travel velocity v is the more relevant quantity since L/v is the time an average traveller needs for a distance L. It is also the quantity which is relevant for fluid-dynamical relations, for example $q = \rho\, v$.

27.2.2 Flow

(also **throughput**). This is traditionally the most important quantity, since it is easy to measure (one just has to count the number of passing vehicles at a fixed location), and it is important for the performance of the transportation system as a whole. In order to allow comparison, it is often useful to divide flow by the number of lanes. Say that during time T we have measured N_T vehicles. Flow then is

$$q_T = \frac{N_T}{T\, N_{lanes}} \; . \tag{27.4}$$

A typical unit of flow is "(number of) vehicles per hour and lane".

Transportation science also uses the term **volume**. According to Gerlough and Huber (1975), this should be reserved to hourly flows (i.e. measured over one hour and expressed in "vehicles per hour"). Maximum flow is also called **capacity**.

There is no direct way to measure space-mean flow. However, sometimes it is useful to use the relation $q = \rho v$. We then have

$$q_L = \rho_L\, v_L = \frac{1}{L\, N_{lanes}} \sum_{i=1}^{N_{veh}} v_i \tag{27.5}$$

where ρ_L is taken from the next section.

[2] Assume that $(v_i)_i$ is a sequence of speed measurements of different vehicles for the space-mean speed. The probability of a vehicle of velociy v_i to cross a sensor within a given time period is proportional to v_i. Thus, in order to obtain spot speed from $(v_i)_i$, each v_i has to be weighted by $w_i = v_i$:

$$v_{spot} = \frac{\sum w_i v_i}{\sum w_i} = \frac{\sum v_i^2}{\sum v_i} = \frac{\sum (v_i^2 - \overline{v}^2) + \sum \overline{v}^2}{\sum v_i} = \frac{N\,\sigma^2 + N\overline{v}^2}{N\overline{v}} = \overline{v} + \frac{\sigma^2}{\overline{v}} \; , \tag{27.3}$$

where σ is the variance of the velocity measurement. This confirms that spot speed is larger than space-mean speed, and the difference increases with increasing velocity fluctuations. – An alternative derivation is, for example, in (Gerlough and Huber, 1975).

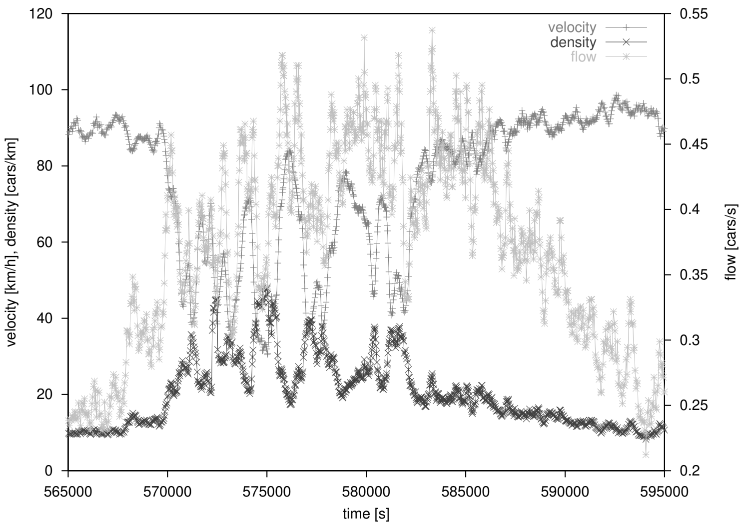


Figure 27.1: Time series of speed, flow, and density.

27.2.3 Density

Space-averaged density ρ_L is the number of vehicles on a certain stretch of road, divided by the length L of that stretch. In order to allow comparison, it is useful to also divide by the number of lanes:

$$\rho_L = \frac{N_{veh}}{L\, N_{lanes}} \ . \tag{27.6}$$

The resulting density is for example given in "(number of) vehicles per km and lane".

Point density has no natural measurement. One can use $\rho_T = q_T / v_T$.

An alternative method for point density is the "fraction of time that a sensor is covered by a vehicle", also called **occupancy**. Unfortunately, this quantity is difficult to obtain from a time-discrete simulation. Since the duration a sensor is covered by a vehicle is ℓ_i / v_i, the correct measurement in a simulation would be **[[check]]**

$$\rho_T = \frac{1}{T} \sum \ell_i / v_i \ . \tag{27.7}$$

In the CA context, $\ell_i = const = 1$. In field measurements, it is usually impossible to obtain ℓ_i for each vehicle, which means that an exact translation of occupancy into density is impossible.

27.3 Fundamental diagrams

As already stated in Sec. 17.3, often speed, flow, and density are not simply plotted as time series, but the relations between them are plotted as so-called fundamental diagrams. Typical fundamental diagrams are speed or flow as the function of density or occupancy. Fig. 27.2 shows the fundamental diagram of flow vs. density obtained from the data of Fig. 27.1. Plausibly, flow is low at low densities (because no vehicle is on the road), and it is low at high densities (because all vehicles are stuck). The behavior in between is however more complex than one maybe would expect, and no complete theoretical explanation is available (Kerner and Rehborn, 1996b; Daganzo et al., 1999; Jost and Nagel, 2003).

[[Some of this will be discussed in Sec. ??]]

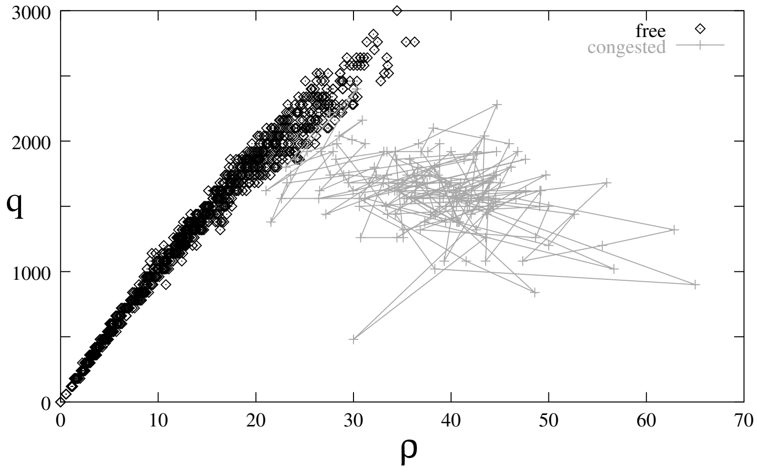


Figure 27.2: Fundamental diagram of flow vs. density from the measurements of Fig. 27.1.

27.4 Car following

27.4.1 Reaction time argument for car following

Any more realistic car micro-simulation first needs to have a method for simple car following. Such methods can be developed on single-lane loops, similar to a single-lane race track. A good way to start is the rule of thumb of "two seconds time headway", that many of us learn at driving school. We are supposed to have two seconds between the time when the car ahead passes a certain location, and the time when we pass it. The reason for this is related to our reaction time. If the car ahead starts braking really hard right when its back bumper is at that location, and if, after a reaction time, we start braking when our front bumper is at that same position, we will barely avoid a crash (see Fig. 27.3). Thus, time headway needs to be larger than reaction time, which translates into a space headway proportional to speed. As a consequence, most car following models have as their most important term one that makes the velocity a roughly linear function of the space headway or gap, although usually a reaction delay of one instead of two seconds is used.[3] All car following models based on this principle have a similar dynamical behavior. For example, the transition from laminar to start-stop traffic is similar for all these models (Krauß et al., 1998). Car following models which are used in micro-simulations are usually designed to be free of accidents.

27.4.2 Discrete space and discrete time: Cellular automata rules

Incarnations of car following can use continuous or discrete time, and continuous or discrete space. While continuous space and continuous time is more realistic, discrete space and time are more natural for a digital computer. And recent research has shown that, in the spirit of Statistical Physics, extremely simple and even unrealistic rules on the microscopic level can still lead to reasonable behavior on the macroscopic level (Krauß, 1997; Nagel, 1996, 1999; Nagel et al., 1998; Brilon and Wu, 1998). In consequence, cellular automata (CA) techniques, which are discrete in space and time, plus have a parallel local update, can actually simulate traffic quite well. They also have a didactic

[3] "Gap" denotes the space from my front bumper to the rear bumper of the car ahead, sometimes minus some safety space one would like to have. Space headway is used less uniformly; for example, it sometimes denotes the front-bumper-to-front-bumper space, thus including the length of the car ahead.

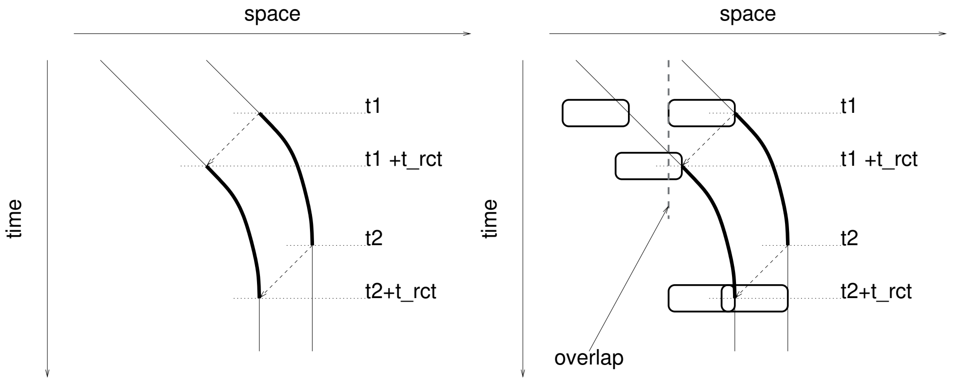


Figure 27.3: Reaction time argument. The left figure shows the trajectories of the front bumpers of two vehicles. At t_1, the leader starts breaking; at t_2, she has come to a standstill. The follower starts breaking at $t1+t_{rct}$; and since his breaking follows exactly the same characteristics, he comes to a standstill at $t_2 + t_{rct}$. The right figure shows the same, with vehicle outlines superimposed. If at $t_1 + t_{rct}$, the follower's front bumper is beyond where the back bumper of the leader was when she started breaking, and accident cannot be avoided (but happens slightly later).

advantage, since coding many aspects of traffic flow such as car following, lane changing, or gap acceptance, is straightforward with a CA approach.

Deterministic traffic CA

As already discussed in Secs. 7 and 17, typical CA for traffic represent the single-lane road as an array of cells of length ℓ, each cell either empty or occupied by a single vehicle. Vehicles have integer velocities between zero and v_{max}. A possible update rule is (Nagel and Herrmann, 1993)

(1) $v_{t+1} = \min[g, v_t + 1, v_{max}]$

(2) $x_{t+1} = x_t + v_{t+1}$

g is the number of empty cells between the vehicle under consideration and the vehicle ahead, and v is measured in "cells per time step".

As will be discussed below, this model has some important features of traffic, such as start-stop waves, but it is unrealistically "stiff" in its dynamics.

As also already discussed in Sec. 17, ℓ is the length a vehicle occupies in a jam, it is often taken as $\ell = 7.5\ m$. In order to get realistic results, a time step of one second is a good choice (remember the reaction time), and then $v_{max} = 5$ corresponding to 135 km/h is a good choice. In applications, v_{max} can be set according to a speed limit on the link. Note that in the traffic CA community distances and speeds are often given without units, which means that they refer to "cells" or "cells per time step", respectively.

This rule is similar to the CA rule 184 according to the so-called Wolfram classification (Wolfram, 1986); indeed, for $v_{max} = 1$ it is identical.

It turns out that, after transients have died out, there are two regimes (Figs. 27.4 and 27.5):

- **Laminar traffic.** All vehicles have gaps of v_{max} or larger, and speed v_{max}. Flow in consequence is $q = \rho\, v_{max}$.

```
.....5.....5.....5....................5.....5..
..........5.....5.....5....................5...
...............5.....5.....5...................
....................5.....5.....5..............

.5.....3...00001.2..3...3...2..3...4....1.01.1.
5.....3...00001.2..3...3...2..3...4....1.01.1.2
.....3...00001.2..3...3...2..3...4....1.01.1.2.
....3...00001.2..3...3...2..3...4....1.01.1.2..
...3...00001.2..3...3...2..3...4....1.01.1.2..3
..3...00001.2..3...3...2..3...4....1.01.1.2..3.
.3...00001.2..3...3...2..3...4....1.01.1.2..3..
3...00001.2..3...3...2..3...4....1.01.1.2..3...
...00001.2..3...3...2..3...4....1.01.1.2..3...4
..00001.2..3...3...2..3...4....1.01.1.2..3...4.
```

Figure 27.4: Space-time plot of deterministic CA. Each line a configuration of the simulated road; traffic goes from left to right; time is going downward. Numbers denote the velocity for the next movement (in cells per time step). TOP: Laminar traffic. BOTTOM: Congested traffic. Some trajectories are added to guide the eye. Note that the *structures* move backwards while the vehicles themselves move forwards. These structures are what the deterministic CA model generates in terms of traffic jams.

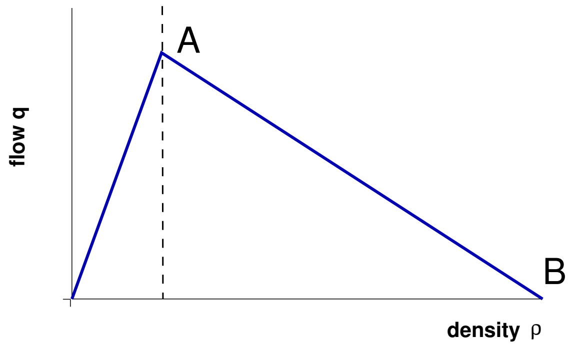


Figure 27.5: Fundamental diagram for the deterministic CA.

- **Congested traffic.** All vehicles have gaps of v_{max} or smaller. It turns out that they allways have a speed equivalent to their gap. This means that $\sum v_i = \sum g_i = N_{veh} \times \langle g \rangle$. Since density $\rho = 1/(\langle g \rangle + 1)$, this leads to

$$q = \rho\, v = 1 - \rho \; . \tag{27.8}$$

The two regimes meet where $\rho\, v_{max} = 1 - \rho$, i.e. at

$$\rho_* = \frac{1}{v_{max} + 1} \; . \tag{27.9}$$

This is also the point of maximum flow, with

$$q_{max} = \frac{v_{max}}{v_{max} + 1} \; . \tag{27.10}$$

Stochastic traffic CA (STCA)

One can add noise to the CA model by adding a randomization term:

(1b) With probability p_{noise} do: $v_{t+1} = \max[v_{t+1} - 1, 0]$; the "max" is needed to prevent negative speeds.

This makes the dynamics of the model significantly more realistic (Fig. 27.6). $p_{noise} = 0.5$ is a standard choice for theoretical work; as already discussed in Sec. 17.3, $p_{noise} = 0.2$ is more realistic with respect to the resulting value for maximum flow (capacity). The stylized fundamental diagram for the STCA looks the same way as the fundamental diagram for the deterministic CA, i.e. as Fig. 27.4. Despite the same shape, the value of maximum flow will however be much lower than with the deterministic CA: about $2000\ veh/hr$ for the STCA with $v_{max} = 5$ and $p_{noise} = 0.2$ (Fig. 17.1) in contrast to $5\ veh/6\ sec = 3000\ veh/hr$ (Eq. 27.10) for the deterministic CA with $v_{max} = 5$.

Slow-to-start rules for STCA

Real traffic may have a strong hysteresis effect near maximum flow; there is however no agreement among researchers if or under which circumstances this effect truly exists. If it exists, it looks as follows: When coming from low densities, traffic stays laminar and at free speed up to a certain density ρ_2 (see Fig. 27.7). Above that, traffic "breaks down" into start-stop traffic. When lowering the density again, however, it does not become laminar again until $\rho < \rho_1$, which is significantly smaller than ρ_2, up to 30% (Kerner and Rehborn, 1996a,b). This effect can be included into the above rules by making acceleration out of stopped traffic weaker than acceleration at all other speeds, for example by:

- if $\Big(v_t = 0$ and $g_t \leq 1\Big)$ then $v_{t+1} = 0$
- else $v_{t+1} = \min[g_t, v_t + 1, v_{max}]$.

This means that the vehicle needs a larger g than before to start moving. Such rules are called "slow-to-start" rules in the physics community (Barlovic et al., 1998; Chowdhury et al., 1999).

Time-oriented CA (TOCA)

A modification to make the STCA more realistic is the so-called time-oriented CA (TOCA) (Brilon and Wu, 1998). The motivation is to introduce a higher amount of elasticity in the car following, that is, vehicles should accelerate and decelerate at larger distances to the vehicle ahead than in the STCA, and resort to emergency braking only if they get too close. For the TOCA velocity update, the following operations need to be done in sequence for each car:

1. if ($g > v \cdot \tau_H$) then, with probability p_{ac},
$$v := \min\{v + 1, v_{max}\}\ ; \tag{27.11}$$
2. $v := \min\{v, g\}$
3. if ($g < v \cdot \tau_H$) then, with probability p_{dc},
$$v := \max\{v - 1, 0\}\ . \tag{27.12}$$

Typical values for the free parameters are $(p_{ac}, p_{dc}, \tau_H) = (0.9, 0.9, 1.1)$. The TOCA generates more realistic fundamental diagrams than the original STCA, in particular when used in conjunction with lane-changing rules on multi-lane streets.

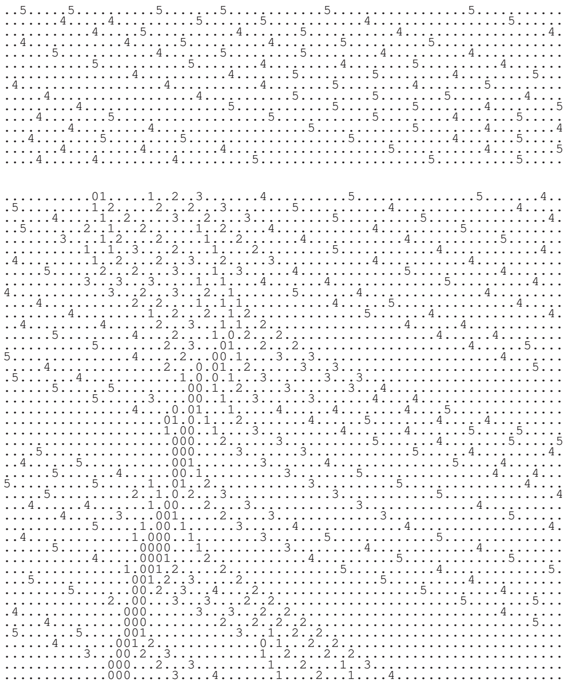


Figure 27.6: Space-time plot of stochastic CA. Each line is a configuration of the simulated road; traffic goes from left to right; time is going downward. TOP: Laminar traffic. BOTTOM: Jam out of nowhere leading to congested traffic.

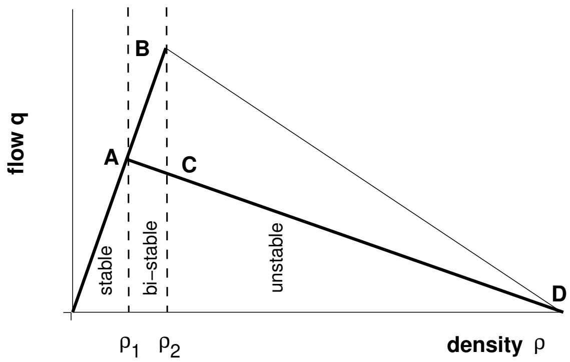


Figure 27.7: Stylized fundamental diagram for slow-to-start STCA.

Dependence on the velocity of the car ahead

It makes sense to assume that velocity difference between vehicles should be included. The idea is that if the car ahead is faster, then this adds to one's effective gap and one may drive faster than without this. In the CA context, the challenge is to retain a collision-free parallel update. Wolf (1999) achieves this by going through the velocity update twice, where in the second round any major velocity changes of the vehicle ahead are included. Barrett et al. (1996) instead additionally look at the gap of the vehicle ahead. The idea here is that, if we know the gap of the vehicle ahead, and we make assumptions about the driver behavior of the vehicle ahead, then we can compute bounds on the behavior of the vehicle ahead in the next time step.

Theory

CA rules can also be analyzed analytically, by means of statistical techniques which look at sequences of configurations of the dynamical evolution of the system (e.g. Schadschneider and Schreckenberg, 1993; Schadschneider, 1998; Chowdhury et al., 2000). Note that this is possible because the cellular approach makes the dynamical states countable: There is only a finite number of possible states for a given number of cells.

27.4.3 Continuous space and continuous time

Making both space and time continuous results in coupled differential equations. Such models for car following were established quite some time ago (e.g. Gerlough and Huber, 1975, and references therein). Most of them also use in one way or other the reaction time argument of Sec. 27.4.1 (as they should). For example, one could use

$$v(t+\tau) = \alpha \, \Delta x(t) \; , \tag{27.13}$$

where Δx is the distance to the car ahead.[4] This just means that, after some time delay, our velocity is proportional to Δx, as it should be according to the reaction time argument.

One can expand $v(t+\tau) = v(t) + \tau \, \dot{v}(t) + ...$, drop second order terms, and rearrange, resulting in

$$\dot{v}(t) = \frac{1}{\tau} \Big(\alpha \, \Delta x(t) - v(t) \Big) \tag{27.14}$$

That is, we adjust our velocity change so that we are adjusting towards the "correct" velocity $v = \alpha \Delta x$. Eqs. (27.13) and (27.14) do not in general generate the same dynamics, in spite of having the same dynamic origin.

A generalization of Eq. (27.14) is to replace $\alpha \, \Delta x_t$ with a function $V(\Delta x(t))$:

$$\dot{v}(t) = \frac{1}{\tau} \Big(V(\Delta x(t)) - v(t) \Big) \tag{27.15}$$

We will need this again later.

[[bando ref]]

The **"classic" car-following model family** (Gerlough and Huber, 1975) comes from taking a time-derivative of the reaction-time relation Eq. (27.13), leading to

$$\dot{v}(t+\tau) = \alpha \Delta v(t) \; . \tag{27.16}$$

After adding some more or less plausible prefactors, this leads to

$$\dot{v}(t+\tau) = \alpha \, \frac{[v(t+\tau)]^l}{[\Delta x(t)]^m} \, \Delta v(t) \; . \tag{27.17}$$

These models are however unstable (e.g. Nagel et al., 2003). The reason behind that is that they allow vehicles to follow each other at extremely close distances with very high speeds as long as there is no velocity difference between them: From $\Delta v = 0$ follows

[4] Car-following models have a tendency to not distinguish cleanly between g (which is space between cars) and Δx (which is usually front-bumper-to-front-bumper distance). As long as vehicles do not pass each other, these differences are indeed irrelevant.

$\dot{v} = 0$. Once a small velocity difference shows up, they react with violent fluctuations. Note that neither Eq. (27.13) nor (27.14) allow such a solution.

For computer implementations, models with continuous time are inconvenient, since time needs to be discretized in one way or other. Because of the reaction delay, many of these car-following equations are delay equations, where considerable effort needs to be spent for faithful numerical results. Given this observation, it seems to be simpler to build models that use discretized time to their advantage (see next section). This is not to say that continuous car-following models are useless; indeed, they continue to contribute to our understanding of the matter (e.g. Bando et al., 1994, 1995). We would expect, however (see below), that any faithful discretization of these equations will run a lot more slowly on a computer than the model presented in the next section, which explicitly uses discrete time.

Another possible implementation of continuous space and time would be event-driven. This works best when particles move with constant velocity for periods of time, interrupted by events where they change it. Molecular dynamics with hard core interactions is an example. Since human driving behavior can probably indeed be characterized like that (Wiedemann, 1994), this should be a promising approach. However, parallel implementations of event-driven simulations are hard and therefore large scale simulations currently not done with this method.

27.4.4 Discrete time and continuous space car following

A disadvantage of the CA approach to traffic is that the coarse-gained description makes fine tuning of many properties difficult. For example, it is difficult to represent fine-grained differences in speed limits, or different acceleration profiles.

On the other hand, the use of coupled ordinary differential equations turns out to be inconvenient for traffic simulations, in particular because of the explizit handling of the reaction time, which means that for numerical integration one needs to maintain the entire dynamical history between t and $t - \tau$ in increments of the time discretization Δt. There are however also models that are continuous in space but coarse-grained discrete in time which work extremely well for traffic (Gipps, 1981; Krauß, 1997; Krauß et al., 1997; Yukawa and Kikuchi, 1995; Sauermann and Herrmann, 1998). The reason for this is that drivers have a reaction delay of about one second, and it is advantageous to use this reaction delay as the time step for the micro-simulation. From a practical point of view, traffic models which use discrete time but continuous space are numerically as efficient as the CA models but are much easier to calibrate. Obviously, a multitude of models is possible here – as is with CAs. We want to concentrate on a single model, a model described by Krauß (Krauß, 1997; Krauß et al., 1997). This model is particularly well understood.

The approach starts again from the reaction time argument (Sec. 27.4.1), this time taking into account the possibility that the two cars can have different velocities. This results in the condition that one's braking distance plus the distance that one drives until one reacts should be smaller than the braking distance of the car ahead plus the space in between the two vehicles. Formally, this yields

$$d(v) + v\,\tau \le d(\tilde{v}) + g \;, \qquad (27.18)$$

where $d(v)$ is the braking distance of a car moving with speed v, τ is the reaction time, g is the distance to the car ahead, and $\tilde{v}$ is the speed of the car ahead ("leader").[5]

[5] Note that this formulation includes the effect of different velocities, but it assumes that acceleration of the follower is zero (?).

Derivation of the safe velocity

Let us first Taylor-expand the function $d(v)$ describing the braking distance around the operating point $\overline{v} := (v + \tilde{v})/2$, where v and $\tilde{v}$ are again the velocity of the follower and leader, respectively:

$$d(v) = d(\overline{v}) + (v - \overline{v})\, d'(\overline{v}) + \frac{(v - \overline{v})^2}{2}\, d''(\overline{v}) + O\Big((v - \overline{v})^3\Big) \ .$$

Inserting this into Eq. 27.18, one obtains first

$$(v - \overline{v})\, d'(\overline{v}) + v\,\tau \le (\tilde{v} - \overline{v})\, d'(\overline{v})\, \tilde{v} + g$$

and then

$$v\, d'(\overline{v}) + v\,\tau \le \tilde{v}\, d'(\overline{v})\, \tilde{v} + g \ . \qquad (*)$$

Note that this is correct up to and including second order, since the second order terms cancel out.

Next, we note the kinematic relation

$$d'(\overline{v}) \equiv \frac{d}{dv} d(\overline{v}) = \frac{\overline{v}}{b(\overline{v})} \ ,$$

where $b(v)$ is the deceleration of the car. This relation can be easily derived when one assumes a constant b until the car is stopped, but is also true for an arbitrary braking profile $b(v)$.

Inserting this into Eq. $(*)$ and rearranging terms yields

$$v \le \tilde{v} + \frac{g - \tilde{v}\,\tau}{\tau + \overline{v}/b(\overline{v})} \ .$$

Showing the collision freeness

In continuous time and after the assumptions made, the above is the condition for collision-free driving. This is true also for the discrete analogue of this formula, provided the step-size h is smaller than τ: First, in general one obtains for the gap

$$g_{t+h} = g_t + h\,\Big(\tilde{v}_{t+h} - v_{t+h}\Big) \ .$$

After using Eq. (27.19) of the main text, rearranging terms, and using the notation $\xi_t := g_t - h\,\tilde{v}_t$ one gets

$$\xi_{t+h} \ge \xi_t \left(1 - \frac{h}{\tau + \overline{v}/b}\right) + h\,\tilde{v}\,\frac{\tau - h}{\tau + \overline{v}/b} \ ,$$

a map $\xi_t \to \xi(t+h)$. Thus, $h \le \tau$ is a sufficient condition to ensure that if $\xi_t \ge 0$, then $\xi t + h \ge 0$, meaning that $\xi_t \ge 0$ for all t if $\xi_{t=0} \ge 0$. Because of the definition of ξ, this ensures that $g_t \ge 0$ for all $t \ge 0$.

Figure 27.8: Derivation of the model by Krauss.

This can be used to derive (see Fig. 27.8) a simple update scheme for the dynamical state of a car:

$$v_{\text{safe}} = \tilde{v}_t + \frac{g_t - \tilde{v}_t \tau}{\overline{v}/b + \tau} \qquad (27.19)$$

$$v_{\text{des}} = \min\{v_t + a\,h, v_{\text{safe}}, v_{\max}\} \qquad (27.20)$$

$$v_{t+h} = \max\{0, v_{\text{des}} - \epsilon\, a\, \eta\} \qquad (27.21)$$

$$x_{t+h} = x_t + h\, v_{t+h} \ . \qquad (27.22)$$

$\overline{v} = (v + \tilde{v})/2$ is the average velocity of the two cars involved, a is the maximum acceleration of the vehicles, b their maximum deceleration, ϵ is the noise amplitude, and η is a random number following a flat distribution in $[0, 1]$.

The terms can be interpreted as follows:

- The first rule (i.e. Eq. 27.19) can be rewritten as

$$v_{safe} = \alpha\, \frac{g_t}{\tau} + (1 - \alpha)\, \tilde{v}_t \qquad (27.23)$$

with

$$\alpha = \frac{1}{\overline{v}/(b\,\tau) + 1} \,. \tag{27.24}$$

[[check on paper]] That is, v_{safe} is a weighted average of g/τ and $\tilde{v}$. For $\alpha < 1$, the velocity of the car ahead is added to the calculation in the following way: If the car ahead is faster, then one can be a little faster than allowed by the gap alone; if the car ahead is slower, then one needs to be slower than allowed by the gap alone.

Note that for $\alpha = 1$ and $\tau = 1$ we recover the STCA rule.

- The second rule (i.e. Eq. (27.20)) just states that the velocity is limited by the desired acceleration a, by the safe velocity v_{safe} as calculated above, and by the maximum velocity v_{max}.

  Note that this is the same as the CA rule.

- In the third term, noise η is added by randomly making the vehicle slower than so far calculated. η denotes a random variable between zero and one, ϵ is a noise scaling factor.

  Again, this is the same as the CA rule.

- The fourth term denotes the forward movement.

For $h \leq \tau$ one can show that this model is free of collisions (Fig. 27.8); normally, one uses $h = \tau$. Typical values for (a, b, ϵ) are $(0.2, 0.6, 1)$.

27.5 Kinematic waves and fluid-dynamics

27.5.1 The Lighthill-Whitham-Richards equation

The intuition for kinematic waves is easy to understand. Start with five vehicles of velocity zero in five adjoining cells. In the first time step, only the first vehicle can move. In the second time step, the second vehicle can start, etc. However, in the meantime it can happen that another vehicle joins the queue at the tail.

Given the right conditions (more vehicles joining at the tail than leaving at the head), this results in a cluster of vehicles of velocity zero and that cluster will move against the traffic direction. Note that the vehicle composition of this cluster is constantly changing – from the perspective of a driver, you join the jam from the end, the jam "moves through you", and then you can start again (look at the two trajectories in the lower part of Fig. 27.4 for an illustration). This is a standard wave phenomenon.

A detailed introduction into such waves can for example be found by Haberman (1977). Here, we will just **[[word?]]** give an overview for people who have some prior knowledge about partial differential wave equations.

One way to see all the connections **[[word?]]** is to start from the standard equation of continuity, which needs to be fulfilled as long as our traffic obeys mass conservation (no vehicles leaving or joining). This equation is

$$\boxed{\partial_t \rho + \partial_x q = 0} \tag{27.25}$$

(**equation of continuity**). This equation can be easily understood when it is discretized (with discretization constants $\Delta t = 1$ and $\Delta x = 1$):

$$N_{t+1}(x) = N_t(x) - \Big(q_t(x+\frac{1}{2}) - q_t(x-\frac{1}{2})\Big) = N_t(x) + q_t(x-\frac{1}{2}) - q_t(x+\frac{1}{2}) \tag{27.26}$$

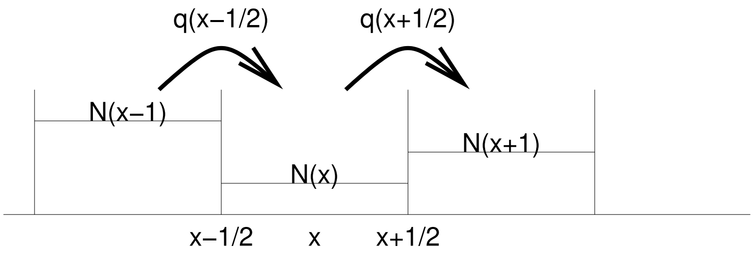


Figure 27.9: Illustration of Eq. (27.26).

where $N_t(x)$ is the number of vehicles in a spatial interval of size $\Delta x = 1$. The notation mirrors the computational implementation, where the spatial index would be represented by an array index, while the temporal index would typically not show up at all. The equation states that the number of vehicles at time $t+1$ is equal to the number of vehicles at time t, plus what flows in from the left, and minus what flows out to the right.

We now need a relation between q and ρ. Let us assume that q is a function of ρ only, i.e. the total differential is $dq = \frac{dq}{d\rho} d\rho$. The meaning of this (instantaneous velocity adaptation) will be discussed below. The resulting theory is also called the **Lighthill-Whitham-Richards (LWR) theory** (Lighthill and Whitham, 1955). **[[Richards ref]]** The equation of continuity can immediately re-written as

$$\boxed{\partial_t \rho + \frac{dq}{d\rho}(\rho)\, \partial_x \rho = 0} \tag{27.27}$$

(**LWR equation**), where $q(\rho)$ is some externally given but as of yet unspecified function.

27.5.2 Linearization

Since we now have a fully defined partial differential equation, we can try to understand some of it. A typical first step is "linearization". For this, ρ is replaced by $\overline{\rho} + \rho'$, with $\partial_t \overline{\rho} = 0$ (stationary) and $\partial_x \overline{\rho} = 0$ (homogeneous); this is always possible. One now *assumes* that ρ' is small. Functions in ρ are Taylor-expanded:

$$F(\rho) = F(\overline{\rho}) + \rho' \frac{dF}{d\rho}(\overline{\rho}) + ... ; \tag{27.28}$$

in our case, we need $F = dq/d\rho$. This results in

$$\partial_t \rho' + \Big(\frac{dq}{d\rho}(\overline{\rho}) + \rho' \frac{d^2 q}{d\rho^2}(\overline{\rho}) + \dots \Big) \partial_x \rho' = 0 \,. \tag{27.29}$$

Finally, higher-order terms (i.e. which contain products of ρ') are dropped, resulting in

$$\partial_t \rho' + \frac{dq}{d\rho}(\overline{\rho})\, \partial_x \rho' = 0 \,. \tag{27.30}$$

This is now a linear equation in ρ', since in each term ρ' occurs at most once. In such cases, one knows that one can make the ansatz

$$\rho' = A\, e^{i(\omega t - kx)} \,. \tag{27.31}$$

If one has never seen this before, it is probably impossible to explain this in two minutes.[6] Inserting Eq. (27.31) into Eq. (27.30) leads to

$$\omega - \frac{dq}{d\rho}(\overline{\rho})\, k = 0 \tag{27.33}$$

[6] There are several elements:

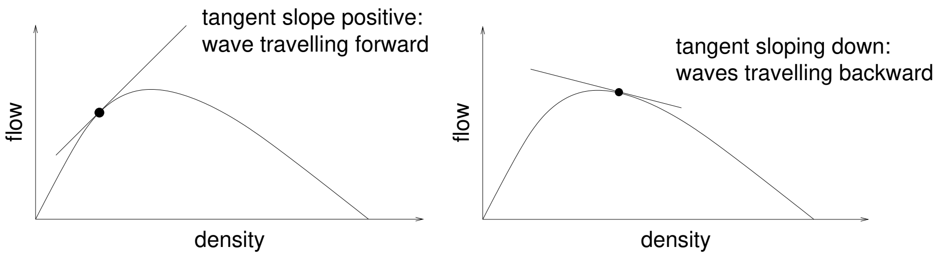


Figure 27.10: Phase speeds of kinematic waves

and therefore to

$$\boxed{c := \frac{\omega}{k} = \frac{dq}{d\rho}(\overline{\rho}) \ .} \tag{27.34}$$

This is the **phase velocity** of the travelling wave. That is, this wave will travel in traffic direction when $q(\overline{\rho})$ is increasing ($\frac{dq}{d\rho}(\overline{\rho})$ positive), and against the traffic direction when $q(\overline{\rho})$ is decreasing (Fig. 27.10).

27.5.3 Macroscopic shocks

Linearization is not very useful for traffic, since it assumes small ρ', which is often not fulfilled in traffic. Let us thus look at a macroscopic front with speed c. Let us go to the same reference system as the front. In that reference system, the flow to the left of the front needs to be the same as the flow to the right of the front, because otherwise there would either be an excess or a lack of "material" at the front. Let us denote variables in the reference system of the front with a tilde. In equations, the statement means

$$\tilde{q}_l = \tilde{q}_r \ . \tag{27.35}$$

Now $\tilde{q} = \rho \, \tilde{v}$, where the density ρ does not need a tilde because it is independent from the speed of the reference system. That is, one has

$$\rho_l \, \tilde{v}_l = \rho_r \, \tilde{v}_r \ . \tag{27.36}$$

For the translation into a non-moving coordinate system, one has $\tilde{v} = v + c$, and therefore

$$\rho_l \, (v_l + c) = \rho_r \, (v_r + c) \tag{27.37}$$

Rearranging yields

$$\boxed{\frac{\rho_l v_l - \rho_r v_r}{\rho_l - \rho_r} =: \frac{\Delta q}{\Delta \rho} = c \ .} \tag{27.38}$$

One can see geometrically that this is just the slope of the line connecting the corresponding points on the fundamental diagram (Fig. 27.11).

- The notation using the complex number i essentially means an equation of type
$$\rho' = A \, \cos(\omega t - kx) \ . \qquad (*) \tag{27.32}$$
What is missing in this simplification is the so-called phase information.
- Eq. $(*)$ is a wave equation. As one can easily verify, it has wave length $2\pi/k$, that is, the function is periodic under additions of $2\pi/k$ to x. k is called the wave number. Similarly, the function is periodic under additions of $2\pi/\omega$ to t; ω is called the frequency.
- One can also verify that, say, a wave crest travels with velocity $c := \omega/k$. In Eq. $(*)$, at time $t = 0$ there is a wave crest at position $x = 0$. At time t, the wave crest is where $\omega t - kx = 0$, which means a velocity $x/t = \omega/k$.

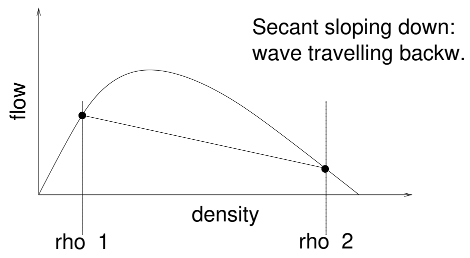


Figure 27.11: Speed of discontinuous fronts

27.5.4 The deterministic CA in terms of kinematic waves

We can now analyse our deterministic CA (Sec. 27.4.2) in terms of kinematic waves (see also Fig. 27.5):

- In the laminar regime, we have $dq/d\rho = v_{max}$. This means that our waves have the same speed as the traffic — that is, they are the "clusters" or "platoons" of cars.
- In the congested regime, $dq/d\rho = -1$. This can be seen in the space-time diagram via the fact that the "patterns" move backwards one cell in each time step (Fig. 27.4 bottom).
- With respect to our introductory problem with the five cars: The jam has density $\rho = 1$ and speed $v = 0$, thus also $q = 0$. Outflow from the jam is eventually at $v = v_{max}$ and $\rho = 1/(v_{max} + 1)$ (this can be seen by following the dynamics). In consquence,

  $$\frac{\Delta q}{\Delta \rho} = \frac{v_{max}/(v_{max} + 1) - 0}{1/(v_{max} + 1) - 1} = -1 \ . \tag{27.39}$$

  Thus, the downstream front of the jam moves backwards with speed $c = -1$. — One could also have seen that by noticing that the outflow is equal to the maximum flow in this model, and then do the geometric solution similar to Fig. 27.11.

  [[might be good to do xfig here too]]

  The inflow is somewhere on the "laminar" branch of the fundamental diagram. That means that the slope of the line connecting to $(\rho = 0, q = 0)$ is either -1 or less steep. The inflow front thus moves backwards with speed 1 or less — that is, the jam will eventually vanish except when inflow is exactly equal to maximum flow.

One can treat queues at traffic lights similarly. While the traffic light is red, $q_{out} = 0$ and thus the outflow front does not move (which we know since the first car is waiting at the red light). The inflow front moves backwards with $c_{in} = q_{in}/(\rho_{in} - 1)$.

Once the traffic light turns green, the outflow front now moves backwards with -1, while the inflow front keeps moving backwards with c_{in}. The situation remains like that until the outflow front catches up with the inflow front. And if the traffic light turns red before that, one needs to include that effect (Fig. 27.12).

27.5.5 More advanced fluid-dynamical models

The kinematic theory is entirely sufficient to understand the most important theoretical aspects of traffic flow. This section goes a little bit beyond that, by providing an outlook what else could be done.

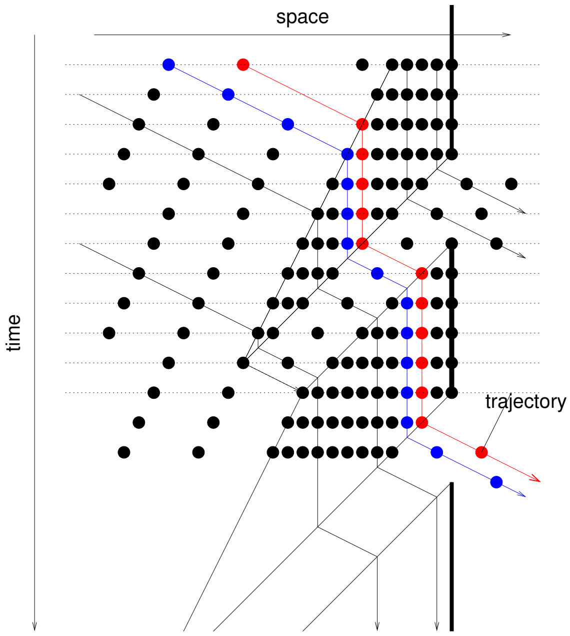


Figure 27.12: Traffic light in terms of kinematic waves

The STCA and in particular the slow-to-start model are not entirely described by the kinematic theory. This is in part due to the stochastic elements, which are not captured in the equation. It is also due to the hysteresis which is displayed by the slow-to-start model (Fig. 27.7) but not by kinematic theory. This motivates to look for fluid-dynamical equations for traffic that capture effects beyond the kinematic theory. Two extensions of the kinematic theory will be discussed.

Addition of diffusive terms

Diffusive terms can be justified for many reasons. The result is an equation like

$$\partial_t \rho + \partial_x q = D \partial_x^2 \rho \; . \tag{27.40}$$

The wave solution after linearization now is **[[check]]**

$$\rho' = A\, e^{-k^2 D t}\, e^{i(\omega t - k x)} \tag{27.41}$$

which means that it has the same phase velocity $c = dq/d\rho$ as before but in addition a decreasing amplitude — waves slowly die out.

Addition of inertia

Above, we have assumed that flow q is a function of the density ρ only. This is in general not true — if a driver suddenly comes into denser traffic, she/he will need some time to adjust; the same is true if density suddenly decreases. That means that velocity will be delayed in its adaptation to density.

A way to capture this is to add an equation for the velocity. One can for example use the car following equation (27.15)

$$a = \frac{Dv}{Dt} = \frac{1}{\tau}\Big(V(\Delta x) - v \Big) \; . \tag{27.42}$$

The translation of the particle-oriented Dv/Dt into the fluid-dynamical $\partial_t v + v\,\partial_x v$ yields

$$\partial_t v + v\,\partial_x v = \frac{1}{\tau}\Big(V(\Delta x) - v\Big)\,. \tag{27.43}$$

We need however $V(\rho)$ instead of $V(\Delta x)$, and we also need ρ measured at the location of the vehicle and not in the middle between two vehicles, where Δx is measured.[7] This is the mathematical reason for what is usually called the **anticipation term**

$$-\frac{c_0^2}{\rho}\,\partial_x \rho\,. \tag{27.46}$$

If density goes up in the driving direction, then $\partial_x \rho$ is positive, thus the term causes negative acceleration, which is plausible.

In addition, we will again add a diffusion term, $\nu\,\partial_x^2 v$. Overall, one obtains the **momentum equation**

$$\boxed{\partial_t v + v\,\partial_x v = \frac{1}{\tau}\Big(V(\rho) - v\Big) - \frac{c_0^2}{\rho}\,\partial_x \rho + \nu\,\partial_x^2 v\,.} \tag{27.47}$$

Note that we still need to specify $V(\rho)$, which is the same information as $q(\rho)$ introduced after Eq. (27.25). The only difference is that we now allow that it can take some time until velocities have adjusted accordingly. Indeed, the relaxation time is τ. If we let τ go to zero, then the momentum equations becomes $v = V(\rho)$, which means instantaneous adaptation.

There is quite a lot of theory about this equation and its meaning for traffic (e.g. Helbing, 1997; Kerner, 1998). Much of the behavior of the micro-simulation models can be explained using these equations; in fact, much of it was first observed in the fluid-dynamical equations. This, however, would be a full class in traffic flow theory and would thus go beyond the scope of this text.

[[breakdown and recovery. do I really want that for this text?]]

27.6 Capacities, especially at bottlenecks

An important concept is **capacity**. The capacity of a link is its maximum flow. As we see from our fundamental diagrams, this looks like a fairly well-defined quantity. For field measurements, a question is which time averages one wants to use. Another question comes up when traffic can "break down", something that we have not discussed in this course.

However, in city traffic, the main obstruction to flow is not the dynamics along the link, but the dynamics at intersections. As an approximate number, an unobstructed link has a capacity of 2000 vehs/hour/lane. If at the end of the link we have a traffic light

[7] Linearization yields

$$V(\rho(\Delta x/2)) = V(\rho(0)) + \frac{\Delta x}{2}\,\frac{dV}{d\rho}\,\partial_x \rho + \ldots \tag{27.44}$$

The second term ("anticipation term") is usually approximated by

$$-\frac{c_0^2}{\rho}\,\partial_x \rho \tag{27.45}$$

in analogy to the sound wave solution of the Navier-Stokes equations. **[[fig for this?]]**

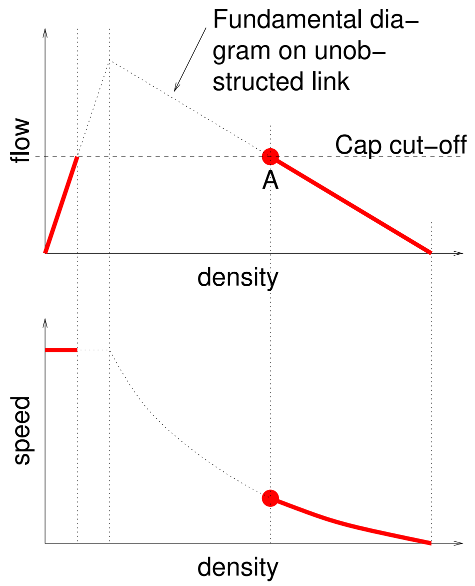


Figure 27.13: Fundamental diagrams when node capacity is smaller than link capacity.

which is green half of the time, then the result will be a link capacity of approximately 1000 vehs/hour/lane. This is a time-averaged number; we have already learned how to describe queue dynamics at traffic lights more realistically via kinematic waves. Here, we will however use the time-averaged description.

If, via the link, there are more cars flowing towards the node than the node can process, then a queue will form. The density inside that queue can be found via the fundamental diagram by going to the high density branch for the given node capacity (point "A" in Fig. 27.13). In consequence, in a situation where the node capacity is smaller than the link capacity, certain density ranges of the fundamental diagram do not occur under steady state conditions.

27.7 Cost-flow curves for static assignment

Traditional models for transportation planning, called "static assignment", do not use any representation of link dynamics at all. The purpose of this section is to explain the traffic dynamics representation of static assignment, and how that relates to the traffic dynamics we have seen so far.

Quite in general, any assignment method needs to be able to calculate link travel times from demand for traffic on a link. Intuitively, travel times increase with demand. The problem seems to be to find a good equation for that – it will however turn out that there is no simple solution.

Static assignment generates steady state solutions. So from a dynamic point of view, **steady state assignment** would be a better name. This means that continuous *streams* of traffic are fed into the system at the origins, and they move via their routes to their destinations, where they are removed. In consequence, demand for a link comes as a flow. So for a simple demand-cost relation we need to find link delay as a function of link flow.

This is actually similar to electricity, where steady-state currents follow an equilibrium pattern through a network according to Kirchhoff's laws. The cost function is Ohm's

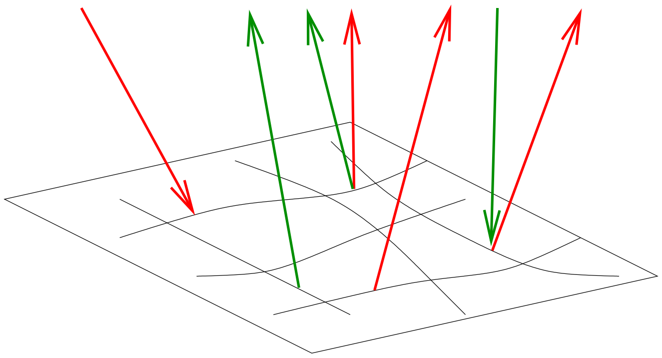


Figure 27.14: Illustration of steady-state network flow.

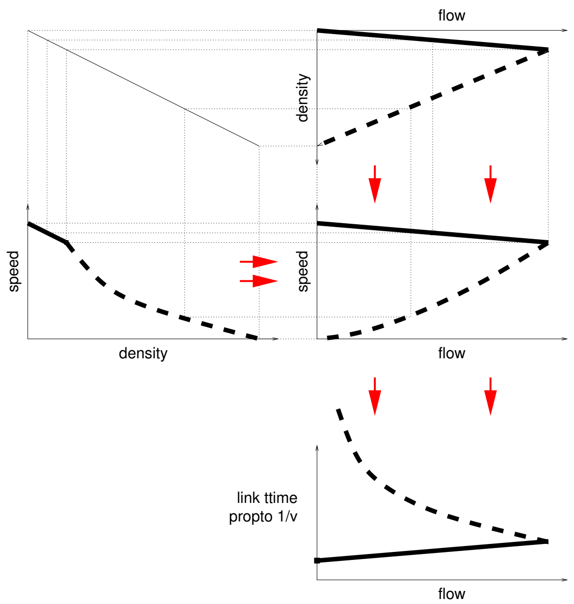


Figure 27.15: Construction of $v(q)$ and thus $T(q)$ for link dynamics. Starting points are the $v(\rho)$ diagram at the left and the $q(\rho)$ diagram at the top.

law, $U = RI$. With constant R, cost is proportional to flow, but R can also depend on I, making this non-linear. The main difference to steady state assignment is that in traffic the particles have fixed destinations which cannot be interchanged.

Now let us construct link travel time as a function of steady state flow for link dynamics. We start from simplified link fundamental diagrams $v(\rho)$ and $q(\rho)$, see Fig. 27.15 left and top, where dashed lines are used in the congested regimes. One can construct or calculate $v(q)$ from that (center right in Fig. 27.15). Link travel time is $T(q) = L/v(q)$; a sketch of this is shown at the bottom of Fig. 27.15.

A problem with this is that there is in general either more than one or no velocity/time value for every given flow value. Looking at the case where the node capacity is the restricting quantity (Fig. 27.16), we see that the problem remains similar for that case. The normal simplification for static assignment has been to only use the upper branch of $v(q)$, which corresponds to the lower branch of $T_{link}(q)$. This results in functions $T(q)$

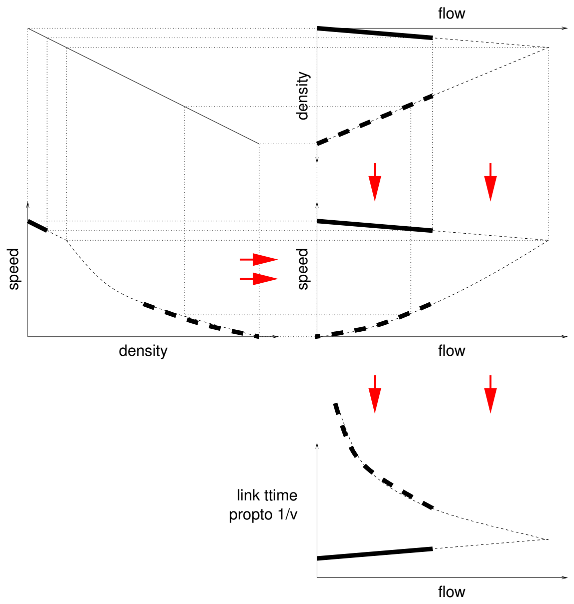


Figure 27.16: Construction of speed and link travel time as function of flow, now for a link with a bottleneck at the end. Inputs are the speed-density relation on the left and the flow-density relation on the bottom.

which in general start at the free speed travel time for zero flow, and which increase with increasing flow, which is plausible.

However, what happens if the assignment model assigns more flow to a link than capacity cap? We know that this is dynamically impossible under steady state conditions. So the only consistent choice for this situation is to set the link travel time to infinity for $q > cap$. This is in fact what static assignment models essentially do, except that they use a smooth function (i.e. no jump at $q = cap$). The main difference between different cost-flow-curves is which cost they give to assigned flows above capacity.

In that sense, it is more reasonable to think about capacity for static assignment as just a free parameter of a cost-flow curve. The calibration of a cost-flow curve is quite difficult, and given the fact that there is no dynamical basis for such a curve, it is clear that it has to be more an art than a science. Nevertheless, the resulting models work quite well, and in spite of knowing better from a theoretical perspective, it is difficult to come up with models that work better in practice.

So far, we have described steady state traffic dynamics and how they are mapped on cost-flow curves for steady state assignment. We have described that one aspect that such models do not pick up are queues upstream of bottlenecks. Note that such queues can well exist under steady state conditions; they violate however the condition that there should only be one velocity/travel time value for each flow value.

There are dynamic aspects of traffic that steady state models cannot pick up at all. A typical scenario is that we have a wide freeway eventually ending in a bottleneck. During rush-hour build-up, the freeway may be used at capacity, resulting in a growing queue at the bottleneck, which will not vanish until the end of the rush period (Fig. 27.17). The steady-state solution would not allow that amount of traffic for the freeway. So here lies one of the reasons why assigment models that are used in practice allow flows above capacity.

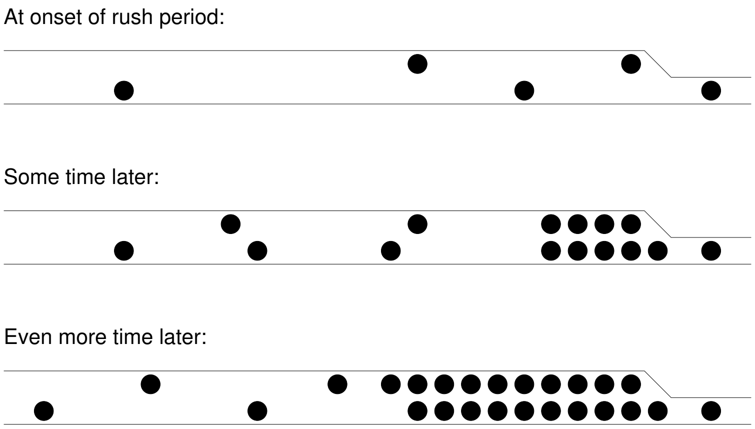


Figure 27.17: A freeway ending in a bottleneck.

There have been attempts to make static assignment models dynamic by solving separate models for several time slices. It is clear that from a dynamical perspective this is not a realistic solution – e.g., the above example with the freeway being used above the bottleneck capacity could still not be picked up.

Chapter 28

Static assignment

[[can I ask Terry (or someone else??) to check this?]]

28.1 Introduction

The traditionally (and currently) most important method for transportation planning is Static Assignment. As said in Sec. 27.7, from our point of view a better word might be Steady State Assignment, since the assumption is that one has constant traffic streams. In fact, the model is very similar to steady state current calculations for electricity or water, where electrons or water molecules enter the system at certain points and are removed at certain other points. The main difference is that for traffic the particles have destinations which they need to reach, which means that in traffic we cannot exchange particles.

This is an extremely basic introduction into static assignment. An introduction at the same level, but with much more material in particular with respect to the history of static assignment, can be found in (Ortúzar and Willumsen, 1995). A comprehensive but still didactic treatment is in (Sheffi, 1985).

28.2 Equilibrium principle

The steady state assignment of electric or water currents to a network follows an equilibrium principle: Along any path through the network, the sum of the voltages is the same. This means that the amount of energy (cost) necessary to go from one point in the network to another one does not depend on the path.

For traffic, the situation is similar, except that our particles have destinations. We thus characterize particles/streams by their (origin,destination) (OD). Only particles which have the same origin and the same destination are treated as interchangeable.

The equilibrium principle is stated as

> *Under equilibrium conditions traffic arranges itself in such a way that no individual trip maker can reduce his/her path costs by switching routes.*

This is **Wardrop's (first) principle**.

If all trip makers perceive the same cost functions, then one can move the point of view from individual travelers to OD flows:

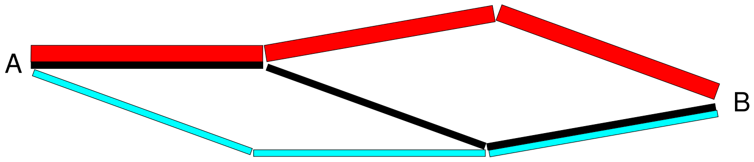


Figure 28.1: Three different path flows connecting A and B.

Under equilibrium conditions traffic arranges itself such that all used routes between an OD pair have equal costs while all unused routes have a cost equal to that or greater.

The idea behind this is: If, for a given OD pair, there is a faster path, then people will start using it, thus making it slower. This process will stop once the new path is as slow as the other paths which are used for this OD pair.

For a mathematical formulation, one needs notation:

- q_a: Flow on link a.[1] $\mathbf{q} = (q_1, q_2, ...)$ is the vector of all link flows.
- $t_a = t_a(q_a)$: Link travel time, as a function of the link flow. *Remember that we have discussed (Sec. 27.7) that such a function does not exist if one looks at the full dynamics. This is the main "problem" with static assignment.*
- Q^{rs} OD flow from r to s (OD matrix).
- There are usually multiple paths p from r to s. $f^{rs,p}$ is the path flow of path p (see Fig. 28.1). In consequence:
$$\sum_p f^{rs,p} = Q^{rs} \, . \tag{28.1}$$
We also reasonably assume that path flows cannot be negative:
$$f^{rs,p} \geq 0 \, . \tag{28.2}$$
- $\delta_a^{rs,p}$ indicates if path rs, p uses link a or not:
$$\delta_a^{rs,p} = \begin{cases} 1 & \text{if used} \\ 0 & \text{if not used} \end{cases} \, . \tag{28.3}$$
- The link flow is the sum of all path flows which use that link (Fig. 28.2):
$$q_a = \sum_{rs,p} f^{rs,p} \, \delta_a^{rs,p} \, . \tag{28.4}$$
- $c^{rs,p}$ is the cost of path rs, p. It is the sum of all link cost contributions:
$$c^{rs,p} = \sum_a t_a \, \delta_a^{rs,p} \, . \tag{28.5}$$

The translation of Wardrop's equilibrium principle into our new notation means that we we are searching for an assignment of the OD streams to the network so that we have
$$c^{rs,p} \begin{cases} = c^{rs} & \text{if path } p \text{ used for } rs \\ \geq c^{rs} & \text{if path } p \text{ not used for } rs \end{cases} \tag{28.6}$$

[1] Conventionally, one uses x here; I will use q because that's what we have used in traffic flow theory.

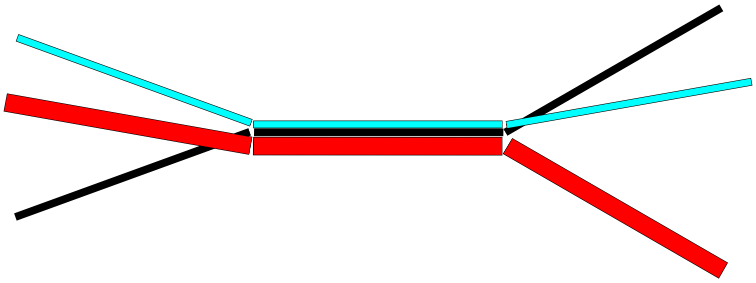


Figure 28.2: A link flow consisting of three path flows.

28.3 Beckmann's mathematical programming formulation

Define a function

$$z(\mathbf{q}) := \sum_a \int_0^{q_a} t_a(\omega)\, d\omega \; . \tag{28.7}$$

The sum is over all links a; for each link, we integrate over the travel time as flow increases, up to the flow q_a actually used on that link.

This is a function which maps high-dimensional space into a scalar number. The number of dimensions is the number of links in the network.

I am not aware of an intuitive motivation for this function. It just turns out that it works: Minimization of this function subject to

$$\sum_p f^{rs,p} = Q^{rs} \quad , \quad f^{rs,p} \geq 0 \tag{28.8}$$

and together with the definitions from above gives the desired equilibrium solution. This is actually not too hard to show. However, the derivation does not give any intuitive insight why $z(\mathbf{q})$ is the correct function.

With this transformation, the equilibrium problem is transformed into a constrained optimization problem. Optimization problems are in general much better understood than equilibrium problems.

28.4 Constrained optimization

Can one provide some intuition of how to solve the problem defined by Eqs. (28.7) and (28.8)? First, ignore the right hand side of Eq. (28.7) and recall that $z(\mathbf{q})$ is just a scalar function in high dimensional space. If $\mathbf{q}$ had only two dimensions, then $z(\mathbf{q})$ could be interpreted as a height function.

The task is to find the global minimum of this function. This is for example similar to finding a global maximum of a fitness function in evolutionary computing.

Since $z(\mathbf{q})$ is analytically given, one can use mathematics to find candidates for global minima. As is known from calculus, all $\mathbf{q}^*$ where $\nabla z\,(\mathbf{q}^*) = \mathbf{0}$ are such candidates. If the problem is constrained, additional candidates are along the boundaries of the allowed regions, see Fig. 28.3. A formal description of this leads to notions such as the **Kuhn-Tucker-conditions** and **Lagrangian multipliers**.

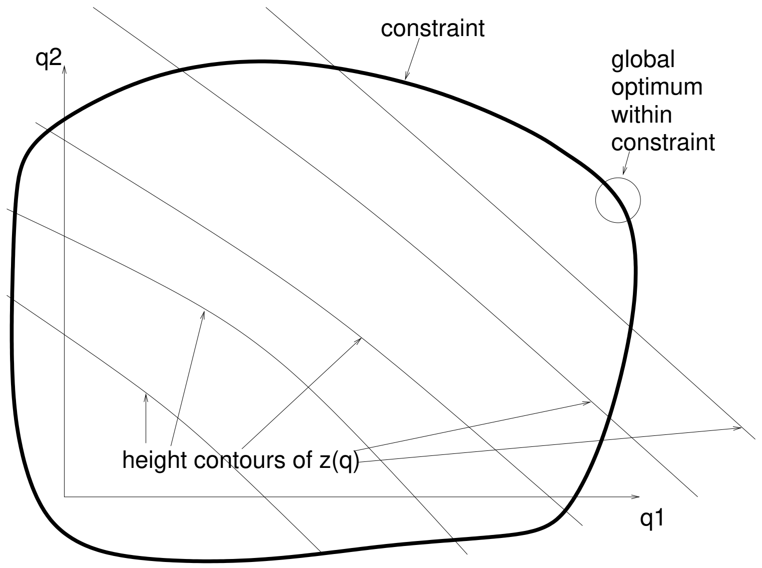


Figure 28.3: Constrained optimization

28.5 Uniqueness

[[Muss das nochmal nachsehen, denn ich denke, dies ist sogar noch st"arker: es gibt nur EIN lokales min, welches damit auch das globale ist.–???]]

One of the major advantages of static assignment is that, under certain conditions, it has one unique solution. This means that no matter what the solution method, all solutions are the same. *This is vastly different from our simulation approach, and certainly one of the big drawbacks of simulation that we have to consider in our work.*

Sufficient conditions for uniqueness of Static Assignment are:

- strict convexity of $z(\mathbf{q})$

together with

- convexity of the feasible region.

These conditions are not minimal, but they are normally used in practice. They will be described in more detail in the following.

28.5.1 Convexity of $z(\mathbf{q})$

Strict convexity of $z(\mathbf{q})$ means, intuitively, that it is "bent" (curved) upwards everywhere. In one dimension, this would be ensured by having a second derivative that is > 0 everywhere. In higher dimensions, it is ensured by having a Hessian ($=$ matrix of second derivatives) that positive definite. A matrix H is positive definite if $\mathbf{v} \cdot H\mathbf{v} > 0$ for all $\mathbf{v} \neq \mathbf{0}$ – this is just the higher dimensional version of "second derivative > 0 everywhere".

For an unconstrained problem, the intuitive interpretation is as follows: Assume there is one location $\mathbf{q}^*$ where $\nabla z\,(\mathbf{q}^*) \equiv \mathbf{0}$, which is therefore a candidate for an optimum. Now if $z(\mathbf{q})$ is curved upwards everywhere, then candidate is a local *minimum*, and there cannot be a second place where $\nabla z(\mathbf{q}) \equiv \mathbf{0}$.

For constrained optimization, one has in addition to make sure that the boundaries cooperate. This is indeed achieved by the convexity of the feasible region, see Sec. 28.5.2.

For Static Assignment, it is possible to simplify the condition of a positive definite Hessian. The calculus for this is a bit tricky, but workable. The result is that the statement

$$H \text{ positive definite} \Rightarrow z(\mathbf{q}) \text{ strictly convex} \tag{28.9}$$

can be replaced by

$$\forall a\colon \frac{\partial t_a(q_a)}{\partial q_a} > 0 \Rightarrow z(\mathbf{q}) \text{ strictly convex.} \tag{28.10}$$

So what we need is that link travel time increases strictly monotonically with link flow. Given the assumptions that we have already accepted, this one is easy to accept.

One has to note that the above will prove convexity of $z(\mathbf{q})$ with respect to the link flows q_a, not with respect to the path flows $f^{rs,p}$. And indeed, the solution is unique with respect to the link flows, but not with respect to the path flows.

28.5.2 Convexity of the feasible region

The feasible region is the set of all solutions which fulfill the constraints. That is, all path flows which fulfill the OD matrix.

Convexity of the feasible region means that any convex combination of feasible solutions is again feasible. A convex combination is a normalized linear combination: If X_1 and X_2 are both feasible, then

$$X_3 := \alpha\, X_1 + (1-\alpha)\, X_2 \tag{28.11}$$

should also be feasible ($\alpha \leq 1$).

[[intuitively, convexity would also avoid "local optima at the boundaries" (see Fig.). This would guarantee convergence of any hillcliming method —???]]

$f^{rs,p} \geq 0$ together with $\sum_p f^{rs,p} = Q^{rs}$ will always result in a convex feasible region.

28.6 A solution method

Constrained optimization is a large area of mathematics, with very sophisticated techniques. Some of these techniques can be used for the static assignment problem (Patriksson, 1994).

Here we want to outline one well-known technique. It is known as Frank-Wolfe algorithm, or convex combinations method. It can be explained in a general way, and then be applied to static assignment, but it can also be applied directly to static assignment, which allows to take advantage of some simplifications right from the beginning. Here we will do the latter.

The idea is to iteratively apply three steps:

1. Linearize $z(\mathbf{q})$ around some operating point $\mathbf{q}^n$, where n denotes the iteration. That is, approximate $z(\mathbf{q}) \equiv z(\mathbf{q}^n + y)$ by

$$z(\mathbf{q}^n) + \mathbf{y} \cdot \nabla z\,(\mathbf{q}^n)\,. \tag{28.12}$$

   The result of this is that the fitness landscape $z(\mathbf{q})$ is replaced by a hyperplane which goes through $z(\mathbf{q}^n)$ and which has the correct slope at $\mathbf{q}^n$.

2. Search, on that hyperplane, for the best solution. On a plane, the best solution is necessarily at the border, so it is sufficient to search the border. Denote this solution by $\mathbf{x}^n = \mathbf{q}^n + \mathbf{y}^n$.

3. Use a convex combination of $\mathbf{q}^n$ and $\mathbf{x}^n$ for a new solution:

$$\mathbf{q}^{n+1} = \alpha\, \mathbf{q}^n + (1-\alpha)\, \mathbf{x}^n \ . \tag{28.13}$$

Ad Item 1: Let us calculate ∇z when applied to $z(\mathbf{q})$ as defined in Eq. (28.7). Let us do that by component, i.e. $(\nabla z)_b \equiv \partial_b \equiv \frac{\partial}{\partial q_b}$. This is the partial derivative with respect to the bth link flow. Only one contribution of the sum depends on q_b at all, and for this one the derivative is trivial:

$$\partial_b \sum_a \int_0^{q_a} t_a(\omega)\, d\omega = \partial_b \int_0^{q_b} t_b(\omega)\, d\omega = t_b \ . \tag{28.14}$$

Therefore, Eq. (28.12) becomes

$$\tilde{z} := z(\mathbf{q}^n) + \sum_a y_a\, t_a(q_a^n) \ . \tag{28.15}$$

Ad Item 2: Eq. (28.15) is maybe a little difficult to interpret at first sight, but it is actually rather straightforward. The task is to minimize $\tilde{z}$ such that the constraints are fulfilled. The constraints are that $\mathbf{q}^n + \mathbf{y}$ fulfills the OD flow conditions. Note that there is no difference if one minimizes $\tilde{z}$ or

$$\hat{z} := \sum_a (\mathbf{q}^n + \mathbf{y})\, t_a(\mathbf{q}^n) \ . \tag{28.16}$$

$\hat{z}$ just means that one has to find feasible flows $\mathbf{x} = \mathbf{q}^n + \mathbf{y}$ such that the sum of all link travel times is minimized, *together with the property that link travel times do not depend on the flows*. This is achieved when every flow takes the fastest path through the network. In other words, $\tilde{z}$ is minimized when OD flows are assigned according to *fastest paths based on the last iteration*.

Interpret that in terms of our agent-based approach: one finds that, given an iteration, progress is made be rerouting some of the OD flows according to what would have been fastest in the last iteration. This is exactly the same in both approaches.

Ad Item 3: The remaining task is to combine the previous solution $\mathbf{q}^n$ and the solution, let us call it $\mathbf{x}^n$, which minimizes $\tilde{z}$. As said above, this is done via a convex combination, i.e.

$$\mathbf{q}^{n+1} = \alpha\, \mathbf{q}^n + (1-\alpha)\, \mathbf{x}^n \ . \tag{28.17}$$

In the agent-based approach, α was just set to 10%, corresponding to a replanning rate of 10%. Because of the analytic formulation in Static Assignment, one can actually search systematically for an optimal α. Alternatively, it is possible to make α dependent on the iteration number via $\alpha^n = 1/n$ (method of successive averages, MSA). For MSA one can prove that the method converges towards the correct solution, although convergence may be slow.[2]

[2] The intuitive reason both for convergence and for slowness is that $\sum_{n=m}^{\infty} 1/n$ always diverges, no matter what m is. This means that any initial contributions to $\mathbf{q}$ can always be fully corrected by later iterations. However, it is also clear that such late corrections take very many iteration steps.

28.7 Summary

The two most important ingredients to static assignment are the assumption of equilibrium and the assumption of steady state, i.e. steady state OD flows. Equilibrium is plausible; and variants of it are currently also used in simulation approaches. The assumption of steady state in contrast leads to the unrealistic distortions of the traffic flow dynamics that we have discussed earlier.

Once these assumptions are made, it turns out that one can formulate the resulting problem as a constrained minimization problem. Under weak additional assumptions (strict monotonicity of the cost-flow-relation), the problem has a unique solution in the link flows. This is a very desirable property, since the solution will not depend on the particular computational method that is used. This is very different from simulation, and certainly an important reason why static assignment is liked so well.

Chapter 29

Discrete choice theory

[[this is not entirely consistent in terms of β, μ, and β_i. Would probably be best to replace the β from the dept time choice by μ.]]

[[this is possibly not entirely consistent in U_X and V_X.]]

[[It might make sense to just teach probit ... would mean however to also replace exp(...) in dept time choice by erf(...).]]

29.1 Introduction

We have seen: Proba to select an alternative A

$$P_A \propto e^{V_A} , \tag{29.1}$$

where V_A utility of option A.

Today: Some formal background.

- Get intuition where functional form e^{V_A} comes from and how other plausible forms can be obtained.
- Learn to interpret coefficient tables ($\rightsquigarrow$ Axhausen).
- Understand how the coefficients are obtained.

Note: Marketing ("toothpaste A or toothpaste B") uses exactly the same technology.

Contents

Binary choice (two alternatives):

- Explain random component.
- Explain choice based on "systematic plus random".
- Understand examples.
- Binary probit or binary logit, depending on distriubtion of randomness.

Multinomial choice (many alternatives). Recover functional form from exercise.

Estimation of the β_i from a survey.

29.2 Binary choice

$=$ choice between two options.

29.2.1 Systematic vs random component of utility

Option A, for example "go swimming".
Has systematic utility (that we compute): V_A.
Assume that (for whatever reason) there is also a random component:

$$\boxed{U_A = V_A + \epsilon_A\ .} \tag{29.2}$$

Choice is made according to U_A.
Possible interpretations:

- Person making the choice is not determinstic.
- Person making the choice is deterministic, but there are additional criteria (for example "was swimming yesterday") which are not included.

  If they were included, then there would be no ϵ_A in this interpretation.

29.2.2 Choice based on random utilities

Now let us assume there are two options, A ("go swimming") and B ("stay home").
We assume that the option with the larger utility is selected (cf. Fig. 29.1):

$$Pr(A) = Pr(U_A > U_B) = Pr(V_A + \epsilon_A > V_B + \epsilon_B) \tag{29.3}$$

$$= Pr(\epsilon_B - \epsilon_A < V_A - V_B) \tag{29.4}$$

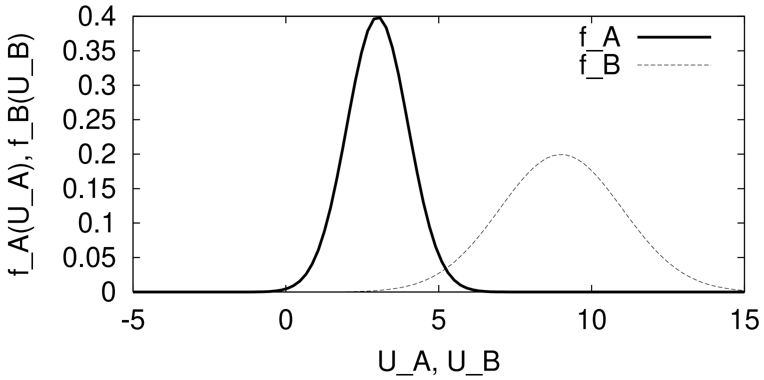


Figure 29.1: Two random distributions, centered around $\langle U_A \rangle = 3$ and $\langle U_B \rangle = 9$. Normally, solution B will win because it has higher utility, but there is a finite probability that U_B will come out really low and U_A comes out really high, in which case A will win.

29.2.3 Linear decomposition of systematic part of utility

Assume that V_A, V_B are linear in contributions:

$$V_A = \beta_1 \, x_{A,1} + \beta_2 \, x_{A,2} + ... = \beta \cdot \mathbf{x}_A \tag{29.5}$$

and similarly

$$V_B = ... = \beta \cdot \mathbf{x}_B \; . \tag{29.6}$$

In principle, the $x_{X,i}$ can be arbitrary functions. In practice, they are usually simple transformations of basic variables, e.g. time, or distance, or distance squared.

29.2.4 Simple example

A result from discrete choice modeling often looks like this:

Car	Bus	Coeff
1	0	-1.4
time with car[min]	time with bus[min]	-0.1
cost with car[cent]	cost with bus[cent]	-0.012

(29.7)

Interpretation: Systematic utility with car is

$$V_{car} = -1.4 - \frac{0.1}{min} \times \text{time w/ car} - \frac{0.012}{cents} \times \text{cost w/ car} \; ; \tag{29.8}$$

systematic utility with bus is

$$V_{bus} = 0 - \frac{0.1}{min} \times \text{time w/ bus} - \frac{0.012}{cents} \times \text{cost w/ bus} \; . \tag{29.9}$$

(Compare: departure time ex.; but this here has only two options.)

For example: Time with car 10min; with bus 20min. Cost with car 200cents; with bus 100cents. Then

$$V_{car} = -1.4 - 1 - 2.4 = -4.8 \; ; \tag{29.10}$$

$$V_{bus} = 0 - 2 - 1.2 = -3.2 \; . \tag{29.11}$$

The probas to select car/bus (see later) will be something like

$$P_{car} = \frac{e^{V_{car}}}{e^{V_{car}} + e^{V_{bus}}} \; . \tag{29.12}$$

$$P_{bus} = \frac{e^{V_{bus}}}{e^{V_{car}} + e^{V_{bus}}} \; . \tag{29.13}$$

29.2.5 2nd example

Car	Bus	Coeff
1	0	-1.4
time with car[min]	time with bus[min]	-0.1
cost with car[cent]	cost with bus[cent]	-0.012
1 if female	0	0.6
1 if (unmarried OR spouse cannot drive OR travels to work w/ spouse)	0	-0.2
1 if (married AND spouse is working AND spouse drives to work indep'y)	0	1.2

Meanings:

If person is female, utility of car is increased.

If person is unmarried OR if spouse cannot drive OR if person travels to work with spouse, then utility of car is decreased.

Etc.

29.2.6 Probability distributions, generating functions, etc.

From this point on, progress is made by making assumptions about the statistical distributions of the noise parameters ε_i. Different assumptions will lead to different models.

Before looking into some specific forms, it makes sense to quickly recall probability distributions and generating functions.

A **probability density function** essentially gives the probability that a certain option is selected. For example, the Gaussian probability density function

$$f(x) = \frac{1}{\sqrt{2\pi}\,\sigma} \exp\left(-\frac{1}{2} \left(\frac{x}{\sigma}\right)^2 \right) \; . \tag{29.14}$$

gives the probability that option x is selected. More precisely, one would have to say that

$$\int_x^{x+\Delta x} f(x) \tag{29.15}$$

is the probability that anything between x and $x + \Delta x$ is selected.

The **generating function** $F(x)$ is the integral of the probability density function. That is

$$f(x) = F'(x) \; . \tag{29.16}$$

In some cases, the generating function is simpler than the probability density function.

The generating function can be used to compute the probability that the selected value is smaller than some given value X. Rather obviously, one has

$$Pr(x < X) = \int_{-\infty}^{X} f(x) = F(X) - F(-\infty) . \qquad (29.17)$$

29.2.7 Binary Probit (Randomness is Gaussian)

Recall: We have

$$Pr(A) = Pr(U_A > U_B) = Pr(\epsilon_B - \epsilon_A < V_A - V_B) . \qquad (29.18)$$

We are now looking for mathematical forms of $Pr(A)$.

Assume that ε_A and ε_B are Gaussian distributed.

Gaussian distributions have the property that sums/differences of Gaussian distributed variables are still Gaussian distributed. In consequence, $\epsilon := \epsilon_B - \epsilon_A$ is Gaussian distributed, for example (with mean zero and "width" σ):

$$f(\varepsilon) = \frac{1}{\sqrt{2\pi}\,\sigma} \exp\left(-\frac{1}{2}\left(\frac{\epsilon}{\sigma}\right)^2\right) . \qquad (29.19)$$

See Fig. 29.2**[[top]]**.

Now we need $Pr(\epsilon < C)$, where $C := V_A - V_B$, and we know that ϵ is normally distributed. As equation:

$$Pr(\epsilon < C) = \frac{1}{\sqrt{2\pi}\,\sigma} \int_{-\infty}^{C} \exp\left(-\frac{1}{2}\left(\frac{\epsilon}{\sigma}\right)^2\right) . \qquad (29.20)$$

[[See Fig. 29.2 bottom.]]

The solution of this needs the so-called error function, sometimes denoted by erf, or `double erf(double x)` under linux. Before the age of electronic computers, the error function was inconvenient to use, which is why the main theoretical development followed a different path, described in the following.

An important piece of knowledge is what happens when random variables are combined. For example, the sum of two Gaussian-distributed random variables are again Gaussian-distributed.

29.2.8 Gumbel distribution

As preparation, learn about the so-called Gumbel distribution:

- Generating function

$$F(\epsilon) = \exp[-e^{-\mu\,(\epsilon-\eta)}] . \qquad (29.21)$$

- Probability denstity function

$$f(\epsilon) = F'(\epsilon) = \mu\, e^{-\mu\,(\epsilon-\eta)} \exp[-e^{-\mu\,(\epsilon-\eta)}] . \qquad (29.22)$$

Location of maximum: η (location parameter).

Variance: $\frac{\pi^2}{6\mu^2} \sim \frac{1}{\mu^2}$ ($\mu =$ width parameter).

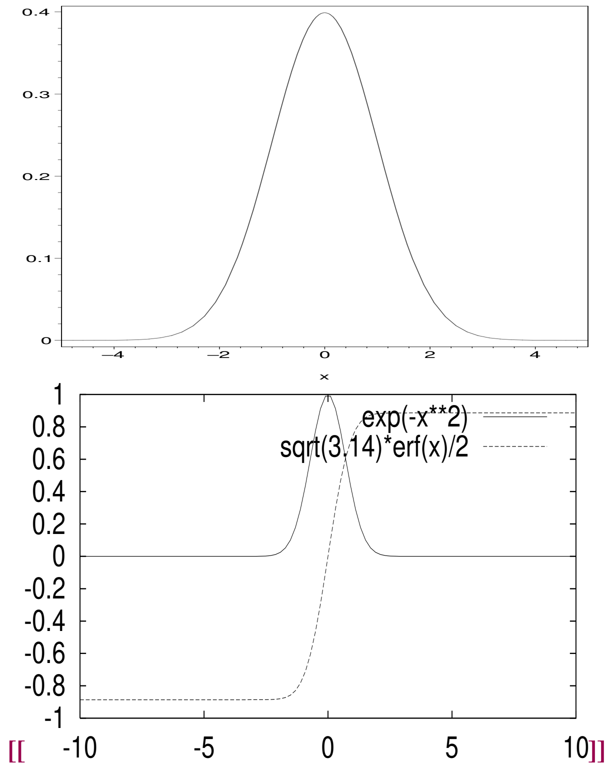


Figure 29.2: **[[TOP:]]** Gaussian distribution. **[[BOTTOM: Error Function "erf", giving the probability that a random variable is larger than x.]] [[this would better be gnuplot]]**

29.2.9 Combination of Gumbel-distributed variables

(Remember: Sum of two Gaussian rnd variables $\rightsquigarrow$ new Gaussian rnd variable with properties ...)

For Gumbel:

- If ϵ_1 and ϵ_2 indep Gumbel with same μ, then $\max(\epsilon_1, \epsilon_2)$ also Gumbel-distributed with the same μ and a new η of

$$\mu^{-1} \ln[e^{\mu\eta_1} + e^{\mu\eta_2}] \,. \tag{29.23}$$

- If ϵ_1 and ϵ_2 indep Gumbel with same μ, then $\epsilon = \epsilon_1 - \epsilon_2$ is logistically distributed (see below) with generating function

$$F(\epsilon) = \frac{1}{1 + e^{\mu\,(\eta_2 - \eta_1 - \epsilon)}} \,. \tag{29.24}$$

29.2.10 Logistic distribution

- Generating function:

$$\boxed{F(\epsilon) = \frac{1}{1 + e^{-\mu\,\epsilon}} \,.} \tag{29.25}$$

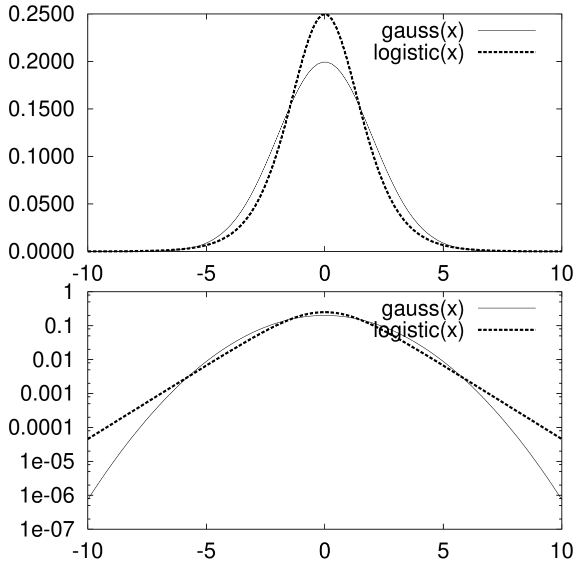


Figure 29.3: Logistic distribution vs. Gaussian distribution, TOP: linear y-axis, BOTTOM: logarithmic y-axis. The logistic distribution is more pointed at its maximum, but has fatter tails (i.e. towards small/large x).

Note that

$$F(-\infty) = \frac{1}{1+e^{\infty}} = \frac{1}{\infty} = 0 \; ; \;\; F(+\infty) = \frac{1}{1+e^{-\infty}} = 1 \; , \tag{29.26}$$

as it should be for a generating function.

- Probability density function:

$$\boxed{f(\epsilon) = \frac{\mu \, e^{-\mu \, \epsilon}}{(1+e^{-\mu \, \epsilon})^2} \; .} \tag{29.27}$$

The logistic probability density function looks somewhat similar to the Gaussian probability density function (Fig. 29.3). μ is the width parameter.

29.2.11 Binary logit (randomness is Gumbel distributed)

Coming back to binary choice, one now assumes that ϵ_A and ϵ_B are Gumbel distributed, meaning that $\epsilon = \epsilon_B - \epsilon_A$ is logistically distributed.

Again, find $Pr(\epsilon < C)$. This is

$$\int_{-\infty}^{C} f(\epsilon) \, d\epsilon = F(C) - F(-\infty) = \frac{1}{1+e^{-\mu \, C}} \; . \tag{29.28}$$

If we re-translate this into our original variables, we obtain

$$\boxed{Pr(A) = \frac{1}{1+e^{-\mu \, V_A + \mu \, V_B}} = \frac{e^{\mu \, V_A}}{e^{\mu \, V_A} + e^{\mu \, V_B}} \; .} \tag{29.29}$$

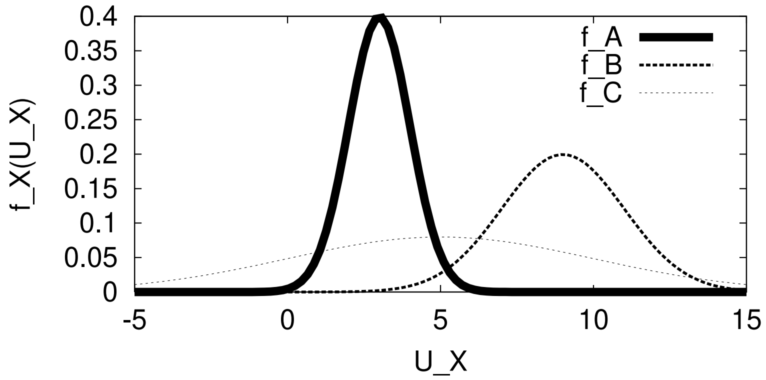


Figure 29.4: Multiple probability density functions for different options. If one picks U_A and U_B, then the probability that C is selected is given by the probability that U_C is larger than the *maximum* of U_A and U_B.

This is similar to what we have seen in the departure time choice (except that here are only two options; for departure time choice we had many).

Note that the noise parameter μ comes from the width parameter of the logistic distribution. Large noise $=$ small μ ($=$ small inverse temperature) $=$ choice more random.

29.3 Multinomial choice

Now more than two choices, e.g.:

- Go swimming, go shopping, stay home, go to movies, ...
- Many possible times-to-depart (discretized into 5-min bins).

See Fig. 29.4.

Concentrate on option "1".

$$P_1 = Pr(U_1 > U_j, \forall j \neq 1) \tag{29.30}$$

$$= Pr(V_1 + \epsilon_1 > V_j + \epsilon_j, \forall j \neq 1) = Pr(\epsilon_j < \Delta V_{1j} + \epsilon_1, \forall j \neq 1)\,. \tag{29.31}$$

Alternatively:

$$P_1 = Pr\left[\epsilon_1 > \max_{j \neq 1}[\Delta V_{1j} + \epsilon_j]\right]\,. \tag{29.32}$$

This is similar to binary choice, i.e. Eq. (29.3). In binary choice, progress was made by assuming that the ε_i were either Gaussian or Gumbel distributed. The same will happen here.

As in binary choice, a Gaussian distribution will lead to use of the error function. This will not be discussed any further here.

A Gumbel distribution will lead to the use of the logistic distribution.

29.3.1 Multinomial logit (MNL)

$=$ multinomial choice with Gumbel-distributed randomness.

We had:

$$P_1 = Pr\left[\epsilon_1 > \max_{j \neq 1}[\Delta V_{1j} + \epsilon_j]\right] . \tag{29.33}$$

Two steps:

1. ϵ_j $(j \neq 1)$ Gumbel-distributed
   $\Rightarrow \epsilon_* := \max_{j \neq 1}[\Delta V_{1j} + \epsilon_j]$ also Gumbel-distributed.
2. ϵ_1 and ϵ_* Gumbel-distributed
   $\Rightarrow \epsilon_* - \epsilon_1$ logistically distributed.

Only problem is to keep track of the transformations of the two parameters η and μ.

Result of second step is (remember: similar to binary logit)

$$\boxed{\frac{1}{1 + e^{\mu\,(V_* - V_1)}} = \frac{e^{\mu\,V_1}}{e^{\mu\,V_1} + e^{\mu\,V_*}} .} \tag{29.34}$$

Either via normalization or via really computing V_* as the new η of the Gumbel distribution one obtains

$$\boxed{= \frac{e^{\mu\,V_1}}{\sum_j e^{\mu V_j}} .} \tag{29.35}$$

29.4 Discussion of modeling assumptions

29.4.1 Independence from irrelevant alternatives (IID)

The multinomial logit model (MNL) predicts that the *ratio* between two options does not depend on other options:

$$\frac{p_i}{p_j} = \frac{e^{\mu\,V_i}}{e^{\mu\,V_j}} . \tag{29.36}$$

There are many cases where this assumption is too strong. The maybe most famous case is the "red bus, blue bus" example. Assume that a traveler has the choice between taking the car, taking a blue bus, and taking a red bus. Assume that the two buses have exactly the same service characteristics; for example, assume that the traveler is the only passenger. Further assume that the probabilities to select the car, the blue bus, and the red bus are 50%, 25%, and 25%, respectively, corresponding to the ratios $2:1:1$. In consequence, the model predicts that the traveler will take her/his car with probability $1/2$.

Now assume that the blue bus is taken out of service. The model now predicts that the ratio between car and red bus will be $2:1$, meaning that the traveler will now take her/his car with probability $2/3$. This is rather implausible since one would assume that the availability of several colors for the bus will not affect the mode choice behavior significantly.

The reason for this behavior can be traced back to the assumption that the ε_i are all statistically independent from each other; this assumption is used when the statistical properties of $\max_j[\Delta V_{1j} + \varepsilon_j]$ and of $\varepsilon_* - \varepsilon_1$ are derived. If they are not statistically independent, then other (usually more complicated) formulations result.

[[the above a little different??]]

29.5 Maximum likelihood estimation

Situation:

- Have survey of $n = 1..N$ persons, and options A, B.
- Also have attributes $x_{n,A,1}$, $x_{n,A,2}$, ... $= \mathbf{x}_{n,A}$ as well as $x_{n,B,1}$, $x_{n,B,2}$, ... $= \mathbf{x}_{n,B}$.

  [This means for example that we know the "time by bus" even if the person never tried that option.]

  Note that we now have a person index n ***everywhere.***
- Also have model specification

$$V_A = \beta_1 \, x_{A,1} + \beta_2 \, x_{A,2} + ... = \beta \cdot \mathbf{x}_A \ . \tag{29.37}$$

How to find $\beta_1, ..., \beta_k$?

29.5.1 ... for binary choice in general

Assume set of persons $n = 1..N$ that were asked.

$y_{n,A} = 1$ means person n chose option A. (Implies that $y_{n,B} = 0$.)

Assuming that we have our model, what is the proba that persons $(1, 2, 3, 4, ...)$ make choices $(A, B, A, A, ...)$? It is (as usual, assuming that the choices are indep)

$$P_{A,B,A,A,...} = P_{1,A} \, P_{2,B} \, P_{3,A} \, P_{4,A} \ ... \ . \tag{29.38}$$

Using the $y_{n,B}$:

$$P_{survey} = \prod_n P_{n,A}^{y_{n,A}} \, P_{n,B}^{y_{n,B}} \ . \tag{29.39}$$

We want, via varying the $(\beta_1, ..., \beta_k)$, to maximize this function.

In words, again: Want high probability that survey answers would come out of our model.

Maximizing in 1d means: Set first derivative to zero, and check that second derivative negative.

Maximizing in multi-d means: Set all first partial derivaties to zero; check that matrix of mixed second derivaties is negative semi-definite.

Instead of maximizing the above function, we can maximize its log (monotonous transformation). Usual trick with probas since it converts products to sums.

$$L = \log P_{survey} = \sum_n [y_{n,A} \, \log P_{n,A} + y_{n,B} \, \log P_{n,B}] \ . \tag{29.40}$$

So far this is general; next it will be applied to Logit.

29.5.2 ... for binary logit model

(Remember: "Logit" means "Gumbel distributed randomness".)

Strategy: Replace $P_{n,X}$ in Eq. (29.39) or in Eq. (29.40) by specific from of logit model, i.e.

$$P_{n,X} = \frac{e^{\beta \cdot \mathbf{x}_X}}{e^{\beta \cdot \mathbf{x}_A} + e^{\beta \cdot \mathbf{x}_B}} \tag{29.41}$$

and then find values β_i such that P_{survey} or L are maximized.

Computer science solution

From a computer science perspective, the maybe easiest way to understand this is to just define a multidimensional function in the variables $\beta_0, \beta_1, \ldots$ and then to use a search algorithm to optimize it.

This function would essentially look like

```
double psurvey ( Array beta ) {
    double prod = 1. ;
    for ( all surveyed persons n ) {

        // calculate utl of option A:
        double utlA = 0. ;
        for ( all betas i ) {
            // utl contrib of attribute i:
            utlA += beta[i] * xA[n,i] ;
        }
        double expUtlA = exp( utlA ) ;

        // calculate utl of option B:
        double utlB = 0. ;
        for ( all betas i ) {
            // utl contrib of attribute i:
            utlB += beta[i] * xB[n,i] ;
        }
        double expUtlB = exp( utlB ) ;

        // contribution to prod:
        if ( person n had selected A ) {
            prod *= expUtlA/(expUtlA+expUtlB) ;
        } else {
            prod *= expUtlB/(expUtlA+expUtlB) ;
        }

    }
    return prod ;
}
```

Search algorithms could for example come from evolutionary computing.

The "computer science" way is almost certainly more computer intensive and less robust than the conventional strategy, lined out next. It does however have the advantage of being applicable also to cases where the conventional strategy fails.

Conventional strategy

The conventional strategy, mathematically more sound but also conceptually somewhat more difficult, is to first invest everything that one knows analytically and only then use computers.

The analytical knowledge mostly involves that one can search for maxima in high-dimensional differentiable functions by first taking the first derivative and then setting it to zero. This is lined out in the following.

Preparations

- Define

$$\boxed{\xi_n = \mathbf{x}_{n,A} - \mathbf{x}_{n,B} \,.} \tag{29.42}$$

In consequence

$$\boxed{P_{n,A} = \frac{1}{1+e^{-\beta\cdot\xi_n}}} \tag{29.43}$$

and

$$\boxed{P_{n,B} = \frac{e^{-\beta\cdot\xi_n}}{1+e^{-\beta\cdot\xi_n}} = \frac{1}{1+e^{+\beta\cdot\xi_n}} \;.} \tag{29.44}$$

(Left version is sometimes useful.)

- First derivative of $\log P_{n,A}$:

$$\frac{\partial \log P_{n,A}}{\partial \beta_k} = -\frac{\partial}{\partial \beta_k} \log(1+e^{-\cdots}) = -\frac{1}{(1+e^{-\cdots})}\, e^{-\cdots}\, (-\xi_{n,k}) \tag{29.45}$$

or

$$\boxed{\frac{\partial \log P_{n,A}}{\partial \beta_k} = \xi_{n,k}\, P_{n,B} \;.} \tag{29.46}$$

Similarly

$$\boxed{\frac{\partial \log P_{n,B}}{\partial \beta_k} = -\xi_{n,k}\, P_{n,A} \;.} \tag{29.47}$$

- We will also need

$$\frac{\partial P_{n,A}}{\partial \beta_k} = (-1)\, \frac{1}{(1+e^{-\cdots})^2}\, e^{-\cdots}\, (-\xi_k) = P_{n,B}\, P_{n,A}\, \xi_k \;. \tag{29.48}$$

Core calculation

Now we can do

$$\frac{\partial L}{\partial \beta_k} = \sum_n \Big(y_{n,A}\, P_{n,B}\, \xi_{n,k} - y_{n,B}\, P_{n,A}\, \xi_{n,k} \Big) \tag{29.49}$$

$$= \sum_n \Big(y_{n,A}\,(1-P_{n,A}) - (1-y_{n,A})\, P_{n,A} \Big)\, \xi_{n,k} = \ldots \tag{29.50}$$

$$= \sum_n \Big(y_{n,A} - P_{n,A} \Big)\, \xi_{n,k} \;. \tag{29.51}$$

When replacing $P_{n,A}$:

$$\boxed{= \sum_n \Big(y_{n,A} - \frac{1}{1+e^{-\beta\cdot\xi_n}} \Big)\, \xi_{n,k} \;.} \tag{29.52}$$

Very good. Now remember that we need to set this, *simultaneously for all* k, equal to zero in order to obtain the values for β which maximize L.

(E.g. Newton in higher dimensions.)

Uniqueness (no contribution to understanding)

Need to check that this is a max (and not a min), and that it is the global max and not a local one.

Reminder: 1d function has max if 1st derivative is zero and 2nd deriv is negative. If 2nd deriv is globally negative, then this is the also the global max.

Translation to higher dimensions: Matrix of 2nd derivatives is globally negative semidefinite.

M negativ semidefinite: $x^T C x > 0$ except for $x = 0$.

Note: Assume $C = M^T M$. Then $x^T M^T M x = (Mx)^T (Mx) > 0$ except for $x = 0$ as long as all entries of Mx are real (i.e. not complex).

Now

$$(\nabla^2 L)_{kl} = \frac{\partial^2 L}{\partial\beta_k \, \partial\beta_l} \sum_n \Big(... \Big) = - \sum_n P_{n,A} \, P_{n,B} \, \xi_{n,k} \, \xi_{n,l} \ . \tag{29.53}$$

Def

$$M_{n,k} = \Big(P_{n,A} \, P_{n,B} \Big)^{1/2} \xi_{n,k} \ . \tag{29.54}$$

Then

$$\nabla^2 L = - M^T \, M \ . \tag{29.55}$$

Since all entries of M are real, $M^T \, M$ is positive definite, and therefore $-M^T \, M$ negativ definite.

29.6 Discussion

29.6.1 The beta parameter from earlier

Sec. 14.3 had used a factor β in front of the utilities, and it was said that smaller β leads to a more random choice, while larger β leads to a stronger preference for the best options. What happened to this β in the theoretical treatment of this chapter?

In fact, the β from Sec. 14.3 is related to the width parameter μ showing up in some equations of this chapter. It is however not systematically treated by this text. The reason for this is that in the maximum likelihood estimation, it does not show up as a separate variable anyway. But what is the reason for this now?

What happens here is that the maximum likelihood estimation automatically includes the meaning of the prefactor β or μ into the other β_i. So if the theoretical form says

$$p_X \propto e^{\mu \, V_X} \tag{29.56}$$

and

$$V_X = \sum_k \beta_k \, x_{X,k} \ , \tag{29.57}$$

then the maximum likelihood estimation in practice estimates the products

$$\tilde\beta_k := \mu \, \beta_k \ . \tag{29.58}$$

The consequence of this is that, if a set of attributes is not useful to predict the choice, then all estimated $\tilde\beta_k$ will be small, leading to quasi-random choice.

[[also: which assumptions were made? Also see in “improvements”]]

29.7 Summary

Foundation: Add random component to systematic utility. We only know systematic component. Assume that max of the sums always wins, which because of random component means that the lower systematic utility sometimes "wins" anyway.

Specific model depends on the distribution function of the random compoment.

Binary choice:

- Gaussian randomness $\rightsquigarrow$ **Binary Probit**. No closed form solution.
- Gumbel randomness $\rightsquigarrow$ **Binary Logit**. Closed form solution $P_A \propto e^{V_A}$.

Multinomial choice:

- Gaussian randomness $\rightsquigarrow$ **Multinomial Probit**. Not treated; no closed form solution. Feasible with computers, and has many theoretical advantages.
- Gumbel randomness $\rightsquigarrow$ **Multinomial Logit (MNL)**. Result again $P_A \propto e^{V_A}$.

Max likelihood estimation of β: Adjust the β so that the probability for the model to generate the survey is maximized.

Chapter 30

Axhausen lecture

Chapter 31

Learning and feedback

31.1 Introduction

In Chap. 22, some pragmatic ways to improve the feedback dynamics were described. This chapter will discuss some background. It will turn out that there are many relations to fixed point relaxation techniques, to Markovian processes, to game theory, and to machine learning. For some aspects, it is possible to provide computational evidence about partial aspects. In general, it however turns out that significant parts of "learning in transportation systems" is a challenging topic where many open questions remain.

31.2 Additional aspects of day-to-day learning

With the exception of Sec. 22.4, we have concentrated on day-to-day learning. Our typical approach is:

1. Generate some initial option for each traveler.
2. Execute that option in the micro-simulation.
3. Allow a certain fraction of the travelers to replace their option with another one, generated by an external module.
4. Goto 2.

In all our implementations, we have suggested to use a randomly selected 10% sample of the population for replanning. Fig. 31.1 shows the effect of different replanning schedules with respect to the sum of all travel times. This figure suggests that all relaxation series relax to the same final result; looking at traffic patterns provides additional support for this statement. There are however important differences in terms of relaxation speed. In particular, runs 4 and 5 were done with a replanning fraction of one percent. Note that in this case, the probability of a traveler never having undergone replanning after 100 iterations is $0.99^{100} \approx 0.366$, more than one third of the population. This is an unacceptably high number, and it explains why even after so many iterations the sum of the travel times is not at the same level as for the others.

All other runs represent higher replanning fractions. Run 1 uses a schedule: 20% replanning in iterations 1–3, 10% replanning in iterations 4–6, 5% in iterations 7–9, and 2% afterwards. Runs 7, 8, and 11 use 5% replanning throughout the iterations, but with a bias towards agents which have not been replanned for a long time. Run 7 in addition

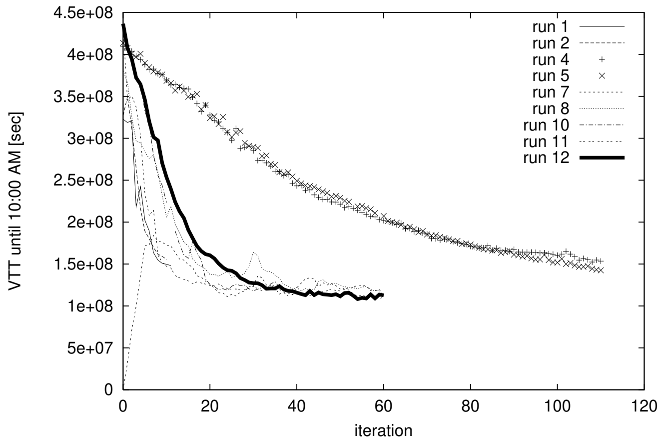


Figure 31.1: Different relaxation paths in day-to-day replanning. The plot shows the sum of all travel times VTT (Vehicle Time Traveled) as a function of the iteration for different relaxation methods. All methods relax to the same value of VTT. From (Rickert, 1998).

loads the network successively, i.e. in the zeroth iteration only 20% of the traffic is put on the network, another 20% is added in the first iteration, etc. Run 10 uses a deterministic instead of a random selection of the travelers for replanning. The advantage is that, with 5% replanning, after 20 iterations one is certain that each traveler was picked exactly once for replanning. In comparison, run 12 uses a simple 5% arbitrary random sample of the population.

The overall result seems to be that, when done right, about 30 iterations are enough to reach relaxation. Also, more complicated selection of agents has no significant advantages over just plain and simple random selection. All simulations refer to the replanning of routes only.

31.3 Individualization of knowledge

31.3.1 Classifier System and Agent Database

Knowledge of agents should be private, i.e. each agent should have a different set of knowledge items. For example, people typically only know a relatively small subset of the street network ("mental map"), and they have different knowledge and perception of congestion. This suggests the use of Complex Adaptive Systems methods (e.g. (Holland, 1992)). Here, each agent has a set of strategies from which to choose, and indicators of past performance for these strategies. The agent normally choses a well-performing strategy. From time to time, the agent choses one of the other strategies, to check if its performance is still bad, or replaces a bad strategy by a new one.

This approach divides the problem into three parts (see also (Ben-Akiva, 2001)):

- Generation of new options. Here new options are generated.
- Evaluation. Here, plans (or strategies) are evaluated. In our context this means that travelers try out all their different strategies, and the strategies obtain scores.
- Exploitation. Eventually, the agents settle down on the better-performing strategies.

As usual, the challenge is to balance exploration (including generation) and exploitation. This is particularly problematic here because of the co-evolution aspect: If too many agents do exploration, then the system performance is not representative of a "normal" performance, and the exploring agents do not learn anything at all. If, however, they explore too little, the system will relax too slowly (cf. "run 4" and "run 5" in Fig. 31.1). We have good experiences with the following scheme:

- A randomly selected 10% of the population obtains new options, and tries them out immediately in the following simulation run.
- All other travelers choose between their existing options, where the probability of selecting option i is taken as
$$p_i \propto e^{-\beta T_i} , \tag{31.1}$$
where T_i is the remembered travel time for that option. β was taken as $1/360\ sec$, which lead (in the scenario that was used) to another 10% of travelers *not* selecting the optimal option.

A major advantage of this approach is that it becomes more robust against artifacts of the router: if an implausible route is generated, the simulation as a whole will fall back on a more plausible route generated earlier. Fig. 31.2 shows an example. The scenario is the same as in Fig. 2.4 of Chap. 2; the location is slightly north of the final destination of all trips. We see snapshots of two relaxed scenarios. The left plot was generated with a standard relaxation method as described in the previous section, i.e. where individual travelers have no memory of previous routes and their performance. The right plot in contrast was obtained from a relaxation method which uses *exactly the same router* but which uses an agent data base, i.e. it retains memory of old options. In the left plot, we see that many vehicles are jammed up on the side roads while the freeway is nearly empty, which is clearly implausible; in the right plot, we see that at the same point in time, the side roads are empty while the freeway is just emptying out – as it should be.

The reason for this behavior is that the router miscalculates at which time it expects travelers to be at certain locations – specifically, it expects travelers to be much earlier at the location shown in the plot. In consequence, the router "thinks" that the freeway is heavily congested and thus suggests the side road as an alternative. Without an agent data base, the method forces the travelers to use this route; with an agent data base, agents discover that it is faster to use the freeway.

This means that now the true challenge is not to generate exactly the correct routes, but to generate a set of routes which is a superset of the correct ones (Ben-Akiva, 2001). Bad routes will be weeded out via the performance evaluation method. For more details see (?). Other implementations of partial aspects are (Unger, 1998, 2002; Gloor, 2001; Weinmann, in preparation).

31.3.2 Individual plans storage

The way we have explained it, each individual needs computational memory to store his/her plan or plans. The memory requirements for this are of the order of $O(N_{people} \times N_{trips} \times N_{links} \times N_{options})$, where N_{people} is the number of people in the simulation, N_{trips} is the number of trips a person takes per day, N_{links} is the average number of links between starting point and destination, and $N_{options}$ is the number of options remembered per agent. For example, for a 24-hour simulation of all traffic in Switzerland, we have $N_{people} \sim 7.5$ mio, $N_{trips} \sim 3$, $N_{links} \sim 50$, and $N_{options} \sim 5$, which results in
$$7.5 \cdot 10^6 \text{ persons } \times \text{ 3 trips per person } \times \text{ 50 links per trip} \tag{31.2}$$

$$\times\ 5 \text{ options} \times\ 4 \text{ bytes per link } = 22.5 \text{ GByte} \tag{31.3}$$

of storage if we use 4-byte words for storage of integer numbers. Let us call this **agent-oriented plans storage**.

Since this is a large storage requirement, many approaches do not store plans in this way. They store instead the shortest path for each origin-destination combination. This becomes affordable since one can organize this information in trees anchored at each possible destination. Each intersections has a "signpost" which gives, for each destination, the right direction; a plan is thus given by knowing the destination and following the "signs" at each intersection. The memory requirements for this are of the order of $O(N_{nodes} \times N_{destinations} \times N_{options})$, where N_{nodes} is the number of nodes of our network, and $N_{destinations}$ is the number of possible destinations. $N_{options}$ is again the number of options, but note that these are options *per destination*, so different agents traveling to the same destination cannot have more than $N_{options}$ different options between them.

Traditionally, transportation simulations use of the order of 1000 destination zones, and networks with of the order of 10 000 nodes, which results in a memory requirement of

$$1\,000 \text{ destinations } \times\ 10\,000 \text{ nodes } \times\ 5 \text{ options per destination } \times\ 4 \text{ bytes per node} \tag{31.4}$$

$= 200$ MByte, considerable less than above. Let us call this **network-oriented plans storage**.

The problem with this second approach is that it explodes with more realistic representations. For example, for our simulations we usually replace the traditional destinations zones by the links, i.e. each of typically 30 000 links is a possible destination. In addition, we need the information time-dependent. If we assume that we have 15-min time slices, this results in a little less than 100 time slices for a full day. The memory requirements for the second method now become

$$30\,000 \text{ links } \times\ 10\,000 \text{ nodes } \times\ 100 \text{ time slices} \tag{31.5}$$

$$\times\ 5 \text{ options } \times\ 4 \text{ bytes per entry} \approx 600 \text{ GByte}\ , \tag{31.6}$$

already more than for the agent-oriented approach. In contrast, for agent-oriented plans storage, time resolution has no effect. The situation becomes worse with high resolution networks (orders of magnitude more links and nodes), which leaves the agent-oriented approach nearly unaffected while the network-oriented approach becomes impossible. As a side remark, we note that in both cases it is possible to compress plans by a factor of at least 30 (Bush).

31.4 Interpretation as dynamical system

We like to interpret our agents and in consequence the whole system as "learning". It is however difficult to exactly define the term "learning"; for example, what is the difference between learning and adaptation? Similarly, it is difficult to formally state the goal of our agents. In the traditional interpretation of economics, reflected in Wardrop's first principle in Chap. 28, agents try to reach a Nash equilibrium, meaning that they are not able to improve by unilaterally changing their strategy. This is however well-defined only within relatively confined formal frameworks and difficult to apply both in complex simulations such as ours and in the real world.

As a first step, it is useful to treat our learning dynamics as a time-discrete dynamical system, and ignore all interpretation. The learning system iterates from one day (period)

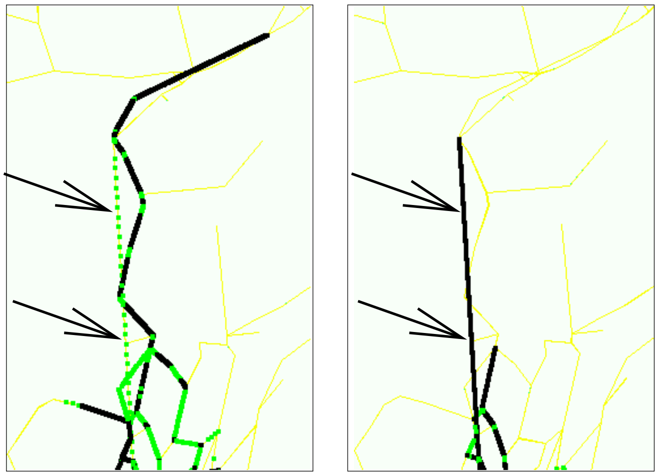


Figure 31.2: Individualization of plans and interaction with router artifacts. LEFT: All vehicles are re-planned according to the same information; vehicles do not use the freeway (arrrows) although the freeway is empty. As explained in the text, this happens because the router makes erroneous predictions about where a vehicle will be at what time. RIGHT: Vehicles treat routing results as additional options, that is, they can revert to other (previously used) options. As a result, the side road now empty out before the freeway. – The time is 7pm.

to the next; a state is all information the system possesses or generates during that day, including agent memory and the trajectory of the simulation through one day; an iteration is the update from one day to the next (Fig. 31.3, although that figure excludes agent memory).

Let us, in order to have some formal symbols at our disposal, denote the state of the system on day n as X_n, and let us denote the operator which maps the system from day n to day $n+1$ as Φ:

$$X_{n+1} = \Phi(X_n) \ . \tag{31.7}$$

This operator subsumes everything that our simulation system does: generation of new options, selection of options, running of the transportation simulation, extraction of scores etc.

[[bottom multi-step iteration?]]

In such a dynamical system, one can search for properties like fixed points, steady state probabilities, multiple basins of attraction, strange attractors, etc. The assumption behind all these concepts is that the system starts out with some arbitrary state, given by the experimentators, but from there on goes to some other state where it will remain.

[[need fig]]

We will assume that our simulations are **Markovian**, meaning that the state at period $n+1$ depends on information from the period n only. If some knowledge about earlier history is involved, then we assume that this is made part of the state at period n. An example for this are the scores of the agents, which contain knowledge from earlier periods. We also assume that the knowledge space of the agents does not infinitely increase, i.e. there is a limit on how many options they remember, and a limit on how much information about the past they remember. For example, when trying the same option several times, the information could be subsumed into a moving average.

Next, we differentiate between deterministic and stochastic systems. Clearly, our transportation simulations are stochastic. Nevertheless, the theory of deterministic dynamic

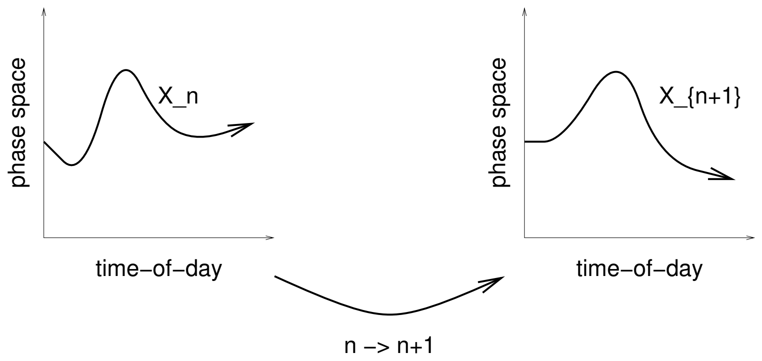


Figure 31.3: Schematic representation of the mapping generated by the feedback iterations. Traffic evolution as a function of time-of-day can be represented as a trajectory in a high dimensional phase space. Iterations can be seen as mappings of this trajectory into a new one. Note that this figure excludes the additional update of agent memory.

systems provides useful insights and often a language to describe what we observe in our systems.

31.4.1 Deterministic systems

It is often of interest to describe the behavior of a system for long times. The following are examples of what can happen. The phenomena do not exclude each other:

- **Fixed point**: A state which repeats itself:

$$X_* = \Phi(X_*) \ . \tag{31.8}$$

  See, for example, Newton iteration in numerical analysis.

- **Periodic behavior**: A cycle which repeats itself:

$$X_{n+k} = X_n \tag{31.9}$$

  for some given k.

- **Chaotic behavior**: Complicated movement, seemingly without rules or structure. Slightly different initial conditions eventually lead to total divergence of the trajectories.

- **Attractor**: A sub-region in state space where the system goes to. Attractors can for example be fixed points, periodic or chaotic.

  A **basin of attraction** is the region of state space which leads to a specific attractor.

- **Ergodic behavior**: The long time trajectory comes arbitrarily close to every point in state space.

Note, for example, that static assignment (Chap. 28) has, under certain conditions, only one optimum. That means that plausible learning dynamics for the static assignment problem have exactly one basin of attraction, and they all lead to the same fixed point solution. This lets us speculate that the result of Sec. 31.2, i.e. that many learning algorithms seem to lead to the same steady state behavior, is caused by structural aspects of the problem, which carry over from static assignment to the simulation variant.

31.4.2 Stochastic systems

In stochastic systems, a state at period n can typically go to more than one state at period $n+1$. This means that in general the notion of a fixed point does not make sense, and needs to be replaced by a **time-invariant probability distribution**. That is, one looks at the probability $p(X)$ for each state X, and how it behaves under our update. Such a probability distribution is time-invariant if

$$p_* = \Phi(p_*) \; . \tag{31.10}$$

Note that this identifies the update operator $\Phi(X)$ for a state with the update operator $\Phi(p)$ for a whole distribution. In stochastic simulation practice, already the computation of $\Phi(X)$ is difficult since it involves running one time iteration over and over again, each time with a different random seed. The computation of a $\Phi(p)$ is normally impossibe and thus useful mostly as a theoretical construct.

Often the words "**in equilibrium**", "**steady-state**", or "**stationary**" are used instead of time-invariant probability distribution.

Again, very little can be said in general about when a system reaches equilibrium. Two conditions which when simultaneously fulfilled lead to convergence to equilibrium are "ergodic" and "mixing":

- **Ergodic:** A system is ergodic if the system can get arbitrarily close to each state from every other state, possibly via a chain of intermediate states.

  [[This definition does not satisfy detailed balance. But I think it is correct. Check!!]]

- **Mixing:** Any initial distribution in state space will spread out and eventually cover the whole state space.

  [[check. In particular, in det Hamiltonian systems, initial phase space vol does not increase, it only fuzzyfies. In stoch systems, this should be different.]]

  What this means intuitively is: Let us start with infinitely many replicas of the same state X_0 but with different random seeds. Being in the same state means that $p(X) = \delta(X - X_0)$. If the system is mixing, then after infinite time the probability to find a randomly picked system in state X is $p_*(X)$, i.e. the steady state density.

In simulation practice, these characterizations are close to useless. Even when a system is both ergodic and mixing, it can display **broken ergodicity**, meaning that it can remain in a part of the state space for arbitrarily long time (Palmer, 1989). For those who happen to know this, a finite size Ising model below the critical temperature is an example. Another example is a stochastic search algorithm being stuck in a local optimum.

[[fig]]

31.4.3 Transients

To make matters worse, we are not necessarily interested in the steady state learning solution, but possibly in the transients. For example, when an important bridge is closed for construction, prediction of the first days after the closure may be as important as prediction of the long term behavior. Worse, aspects such as land use or the housing market in practice probably never reach the steady state.

To put this into context, consider a simple ordinary differential equation,

$$\frac{df}{dt} = -f \; . \tag{31.11}$$

The steady state solution to this can be found by setting $df/dt = 0$, that is, it is $f = 0$. The well-known complete solution is

$$f(t) = f_0 \, e^{-t} \, , \tag{31.12}$$

where f_0 is the initial state. What this means is that we are used to systems where we can describe not only the steady state solution, but also the transients. It is not clear if we will ever reach a similar level of understanding of learning dynamics.

31.5 Relation to game theory

A **Nash Equilibrium** (NE) is a state where no agent can improve its payoff by unilaterally changing its strategy. In terms of this text, this means the system is at a NE if no agent can improve its score by unilaterally selecting a different (routing/activity/...) option. An equilibrium in game theory is a static concept; it is in consequence not the same as an equilibrium in dynamical systems.

For static assignment (Chap. 28), we have seen this as Wardrop's first principle, and the theory of static assignment started from there. We have also seen that in the case of static assignment, under certain conditions the solution was unique, meaning that there was only one NE.

The construct of a NE does not say anything about how a system can reach it. In standard game theory, it is assumed that each agent completely pre-computes its moves and then submits a "strategy book" to the referee, who will then play the game for the agents. The Nash Equilibrium definition implies that the solution is (marginally) stable if exactly one player deviates from the NE. Nothing is said about stability if two players simultaneously deviate from the NE.

Sometimes, a NE is a fixed point of a certain type of deterministic learning dynamics. A typical example is **best reply**, where each player plays what would have been optimal in the last period. If an agent has several best options, it choses the same as in the last period (if applicable). Under best reply, a NE, once reached, is repeated forever. Again, this does not say anything about stability, since fixed points can be attractive ($=$ stable), neutral, or repulsive ($=$ unstable).

There are subtleties involved in a translation from game theory to dynamical systems. Most importantly, one has to assume that in the dynamical system interpretation, the agents do not actively optimize any given quantity beyond the prescription of the dynamics. Rather, their behavior is completely given by the dynamic description, and this dynamics sometimes happens to have the NE as a fixed point. For example, the situation is different if an agent attempts to optimize the average reward over all iterations.

When moving from deterministic to stochastic simulations, the usual changes are necessary. In particular, the NE has to be suitably redefined, for example that each agent should not be able to improve the *expected* reward. Although this sounds feasible in theory, it is difficult in practice, since we do not know how to compute the expected reward via simulation. An approximation to the expected reward would be to simulate the transition from n to $n+1$ with many different random seeds and average over all occuring rewards; however, this is neither computationally efficient nor plausible from the point of view of reality.

In conclusion, it seems that we are left with a system which has some relation to game theory, but they are not exactly the same. It is possible to change our system so that it maps exactly on game theory, but only by moving it farther away from what we would expect as plausible human behavior.

31.6 Relation to machine learning

There is also a connection of our simulations to machine learning. This connection becomes clear if we consider each agent as a learning machine – in consequence, all knowledge from machine learning (which typically considers a single agent in an environment) could be applied to our agents. In other word, each agent could be programmed as a learning machine, using the best of methods available from machine learning. This leads to several issues:

- In how far are machine learning methods applicable under the constraints that we face? In particular, we need to have of the order of 10^7 learning agents, and we have a non-stationary environment (since also the other agents learn).[1]

  On the other hand, very little of what we have considered concerns states being dependent on each other, i.e. the situation faced in reinforcement learning that the expected pay-off has both immediate and long-term contributions. This is however a simplification in transportation that does not truly apply. For example, path finding could also be considered as a state-dependent operation; and weekly activity lists where leisure, shopping, going to the doctor has to be distributed across several days leads to similar issues.

- In how far does the result resemble human learning? In other words, how far different is human learning and machine learning for the questions we are interested in?

- Does our system have anything to do with *distributed* machine learning? That is, can the whole transportation system be considered as a large multi-agent learning system? In contrast to typical approaches in artificial intelligence, there is no obvious goal that the transportation system attempts to optimize.

  In other words: How large is the difference between distributed learning systems for solving a given task, and distributed learning systems as models for human society?

The last aspect also becomes apparent when comparing the concept of a Nash Equilibrium with the concept of a **System Optimum (SO)**. Whereas the first assumes that every agent opimizes its own utility, the latter assumes that some system-wide quantity is optimized. For example, one could optimize the sum of all travel times rather than having each individual agent optimizing its travel time. The results are in general *not* the same; the NE solutions lead to larger travel times.

[[additional section: Gibbs sampling (Markov chain monte carlo)?]]

[[with-day section–??]]

31.7 Smart agents and non-predictability

A curious aspect of making the agents "smarter" is that, when it goes beyond a certain point, it may actually *de*grade system performance. More precisely, while average system performance may be unaffected, system variance, and thus unpredictability, invariably goes up. An example is Fig. 31.4, which shows average system performance in repeated runs as a function of the fraction f of travelers with within-day replanning capability. While average system performance improves with f increasing from zero to 40%, beyond that both average system performance and predictability (variance) of the

[1] More precisely: The agent cannot assume that the probabilities are constant since the other agents also learn. However, in the long run all probabilities will become constant.

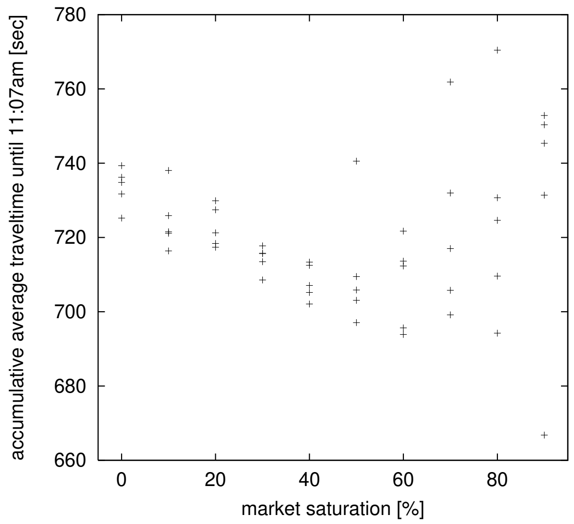


Figure 31.4: Predictability as function of within-day rerouting capabilities. The result was obtained in the context of a simulation study of route guidance systems. The x-axis shows the fraction of equipped vehicles; the y-axis shows average travel time of all vehicles in the simulation. For each value of market saturation, five different simulations with different random seeds were run. When market saturation increases from zero to 40%, system performance improves. Beyond that, the average system performance, and, more importantly, also the predictability (variance) of the system performance degrade. From (Rickert, 1998).

system performance degrade. In other words, for high levels of within-day replanning capability, the system shows strong variance between uncongested and congested. From a user perspective, this is often not any better than bad average system performance – for example, for a trip to the airport or to the opera, one usually plans according to a worst case travel time. Also, if the system becomes non-predictable, route guidance systems are no longer able to help with efficent system usage. The system "fights back" against efficient utiliziation by reducing predictability.

Results of this type seem to be generic. For example, Kelly reports a scenario where many travelers attempt to simultaneously arrive at downtown for work at 8am (Kelly, 1997). In this case, the mechanism at work is easy to see: If, say, 2000 travelers want to go to downtown, and all roads leading there together have a capacity of 2000 vehicles per hour, then the arrival of the travelers at the downtown location necessarily will be spread out over one hour. Success or failure to be ahead of the crowd will decide if one is early or late, very small differences in the individual average departure time will result in large differences in the individual average arrival time, and because of stochasticity there will be strong fluctuations in the arrival time from day to day even if the departure time remains constant. Ref. (Nagel and Rasmussen, 1994a) reports from a scenario where road pricing is used to push traffic closer towards the system optimum. Also in this case, the improved system performance is accompanied by increased variability. Both results were obtained with day-to-day replanning.

31.8 Conclusion

The approach of this class **[[book]]** to agent learning was that the learning method is first described as a computer algorithm, and the behavior of the algorithm is analyzed later.

The first level of analysis is the analysis of the resulting dynamics, without any normative statements. Day-to-day dynamics is discrete in time, and can be analyzed as any time-discrete deterministic or stochastic system. In all generality, this does not help much, since possible outcomes range from fixed points to chaotic attractors; it does however provide a language to describe resulting behavior and to classify what to expect.

In terms of a normative theory, game theory comes in. Our system can be interpreted as all agents attempting to find their best solution, given the behavior of all other agents (Nash Equilibrium). With appropriate care, some versions of a learning dynamics will contain Nash Equilibria as fixed points. The mapping of our learning dynamics into game theory does however move the simulations away from what seems behaviorally plausible.

Third, there are relations to machine learning. In particular, each agent can be seen as a learning machine. The two most important differences to standard machine learning are: We have many more agents, and there is no common goal.

Finally, the chapter has described some examples of where smarter agents lead to larger instabilities. Such examples seem to be generic, also outside the area of transportation. Care needs therefore to be taken to not make simulations and reality more unstable by adding more information.

Part V

Calibration and validation

[[It would make sense to put traffic flow characteristics at the end of improvements and then rename this chapter "real world case studies". However, do we need stuff from "traffic flow theo" in "background" for the traff flow char chapter?]]

Chapter 32

Traffic flow characteristics

32.1 Introduction

One could probably reach agreement that the traffic flow behavior of traffic simulation models should be well documented. Yet, in practice, this turns out to be somewhat difficult. Many traffic simulation models are under continuous development, and the traffic flow dynamics documented in a certain publication is often a "snapshot", valid at the time of writing, but no longer the true state of the model.

It thus makes sense to agree on a certain set of tests for traffic flow dynamics which should be run and documented together with "real" results. In this paper, we propose a (probably incomplete) suite of traffic flow measurements. Also, some of the results in this paper are arguably unrefined with respect to reality. Yet, as we stated above, we are continuously working on improvements, and this publication represents both a snapshot of where we currently stand and an argument for a standardized traffic flow test suite for simulation models. We hope that this publication will both open the way for a constructive dialogue on which standardized traffic flow tests should be run for traffic simulation models, and which of the features of our traffic simulation models may need improvement.

This paper starts with a general section on validation and calibration (Sec. 2), followed by a high-level description of the Transims microsimulation approach (Sec. 3). Sec. 4 is a fairly technical description of the actual implementation. Sec. 5 contains a description of the test cases that we ran for this paper and presents the simulation results. Sec. 6 contains an example of parameter sensitivity testing for the case of a yield sign, followed by a short section outlining differences in the logic when the simulation was used for the so-called Dallas case study (Sec. 7). The paper is concluded by a discussion section and a summary.

32.2 Validation, Calibration, etc.

Prerequisite of any simulation model to be used is a certain amount of confidence in its output. The process of building confidence depends on human nature and is sometimes hard to explain. Yet, an organized process towards model acceptance would help. Such an acceptance process may be composed of the following four elements Van Aerde (personal communication):

- **Verification** – have the hypothesized behavioral rules been implemented correctly?

- **Validation** – do the hypothesized behavioral rules produce correct emergent behavior, such as correct fundamental diagrams? Note that this does not specify a quantitative procedure; plausibility, consistency with theory and experience, and documentation of emergent behavior are the important elements here.
- **Calibration** – have the model parameters been optimized to (possibly site specific) settings? This requires a decision on a data set and a decision on an objective function that can *quantify* the closeness of the simulation to the data set.
- **Accreditation** – Given a question, is the model indeed powerful enough to help with it?

Note that this process is not uni-directional. For example, if one cannot calibrate a model very well for a given scenario and a given objective function, one will go back and change the microscopic rules and then have to go through verification and validation again.

Also, a formally correct verification process can be shown to be mathematically hard or computationally impossible except in very simple situations (see, e.g., Chapters 14 - 16 in Van Leeuwen (1990)). Intuitively, the problem is that seemingly unrelated parts of the implementation can interact in complicated ways, and to exhaustively test all combinations is impossible. For that reason, both practitioners and theoreticians suggest that one needs to allocate resources intelligently between verification and validation.

Sometimes, the word "validation" is also used when a simulation model, *after* calibration to a scenario and data set A, is run under another scenario to test its predictive performance. Since this represents in principle the same procedure – run the simulation model against a scenario without further adjustment in the process – we do not see a problem in the use of the word validation in both cases.

Next, one needs to decide on which networks to run the above studies. The following seem to be useful:

- **Building block cases** such as "traffic in a loop" or "traffic through yield sign". The chapters of the Highway Capacity Manual Transportation Research Board (1994b), despite being under discussion, seem to be a good starting point here. Maybe these cases will not be very useful for calibration since "clean" data on these cases is difficult if not impossible to obtain. Yet, these cases would certainly allow plausibility check of a simulation model, and comparison to other simulation models.
- **Complicated test cases**, which test a variety of behavior such as merging or traffic signals, in a larger *context* (i.e. when interacting). It would be nicest to have test cases from the real world, together with real data. – These test cases would best be made electronically available.

Of course, models have always been validated and calibrated, e.g. Cassidy and Han (1995); Mahmassani et al. (1987); Ponzlet and Wagner (1996). For fluid-dynamical models, calibration can be formalized Cremer and Papageorgiou (1981); Cremer and Schütt (1990). Yet, we would like to stress that there are two diverging tendencies here:

- Models which are simple (i.e. have few parameters) are easy to be formally calibrated in the sense that one can adjust the parameters so that some objective function is minimized. Yet, the model may be too simple to indeed reflect the "meaning" of the data.[1]
- Models which have many parameters are in principle capable of representing a much wider variety of dynamics. Yet, they are difficult for formal calibration because the degrees of freedom are too large. Here, the intuition of the developer

[1] Bluntly, one can always fit a straight line to a data cloud.

is important, who prescribes the simplifications, usually by making the problem more homogeneous than it is (for example prescribing that drivers only fall into few behavioral classes). – Microscopic models fall into this category.

Ref. Denney et al. (1993) nicely illustrates the problem: The authors indeed decide on an objective function (match the two parameters of a two-fluid model description of the real world traffic); yet the procedure is trial and error in the sense that the authors themselves decide on which aspects of NETSIM they believe to be important.

This indicates, consistent with our own experience, that formal calibration (in the sense of a formal procedure as opposed to trial-and-error) of *microscopic* models is currently very hard to achieve. This, in addition to the generally valid argument that calibration does not protect one against having the wrong model, implies to us that on the "validation" level, comparable and meaningful test suites should be constructed, and that the model behavior in these test suites should be publicly documented. This effort should be geared towards *understanding* the strength and weaknesses of a/the participating model (as opposed to deciding which is the "best" model).

In this paper, we want to concentrate on the "validation" part in the above sense in conjunction with "building block" test cases. We mean that as a first important step; in the future, we would like to be able to say something like "the simulations in this study are based on driving rules with their emergent behavior documented in the appendix", which would recognize the fact that the rules may have changed since the last "major" publication. This does not preclude that we will attempt to construct more realistic test scenarios in the future.

32.3 The Transims microsimulation approach

When designing a traffic microsimulation model, the first idea might be to measure all aspects of human driving and put them in algorithmic form into the computer. Unfortunately, such attempts cause many problems. The first is a data collection problem, because one can certainly not measure "all" aspects of human driving and is thus faced with the double sided problem that the necessary data collection process is extremely costly and still selective. Second, what if the emergent flow properties of such a model are clearly wrong, for example producing an hourly flow rate that is much too high?

For that reason, the Transims (TRansportation ANalysis and SIMulation System TRANSIMS www page (accessed 2003)) microsimulation starts with a *minimal* approach. A minimal set of driving rules is used to simulate traffic, and this set of rules is only extended when it becomes clear that a certain important aspect of traffic flow behavior cannot be modeled with the current rule set.[2] Besides the conceptual clarity, this also has the advantage that it is usually computationally fast – minimal models have few rules and thus run fast on computers.

The last paragraph leaves open what the "important aspects" are. In our view, this can only be decided in the proper context, i.e. when the question or problem area of application is known. The questions that Transims is currently designed for are transportation *planning* questions. These questions have traditionally been approached using traffic assignment models based on link performance functions (link capacity functions). Link performance functions are known to be dynamically wrong in the congested regime Patriksson (1994); they simply do not model queue build-up when demand is higher than capacity.

The most important result of a transportation microsimulation in that context should be the *delays*, since they dominate travel times, and also hinder discharge of the transporta-

[2]Note, though, that it is certainly desirable to have *reasonable* microscopic rules.

tion system, thus leading to grid-lock. Delays are caused by congestion, and congestion is caused by demand being higher than capacity. This implies that the first thing the Transims traffic microsimulation has to get right are capacity constraints (and possibly their variance). Capacity constraints are caused by a variety of effects:

- Undisturbed roadways such as freeways have capacity constraints given by the maximum of the flow-density diagram.
- Typical arterials have their capacity constraints given by traffic lights.
- In the case of unprotected turning movements (yield, stop, ramps, unprotected left, etc.), the capacity constraints are given as a function of opposing traffic flows. For example, the number of vehicles making an unprotected left turn depends on the oncoming traffic.

Building a simulation which can be adjusted against all these diagrams seems a hopeless task given the enormous amount of degrees of freedom. The Transims approach for that reason has been to *generate* the correct behavior from a few much more basic parameters. The correct behavior with respect to the above criteria can essentially be obtained by adjusting two parameters: (i) The value of a certain asymmetric noise parameter in the acceleration determines maximum flow on freeways and through traffic lights; (ii) the value of the gap acceptance determines flow for unprotected movements.

It needs to be emphasized again that these remarks are only valid in our context: There are many questions for which the models need to have a higher fidelity, and then more details, higher resolution, etc. may need to be added (e.g. Wiedemann (1994); Van Aerde et al. (1996)).

There is sometimes debate whether the model we thus obtain is truly "microscopic". We use the term "microscopic" with respect to the *resolution* of the model, i.e. a model is microscopic as soon as it allows the identification of individual particles (here cars). The proposed area of *application*, though, is where traditionally more macroscopic models have been used Patriksson (1994); Chang et al. (1985); Schwerdtfeger (1987); Herman and Prigogine (1979).

32.4 Rules of the model

32.4.1 Single lane uni-directional traffic

Our traffic simulation is based on a cellular automata technique, i.e., a road is composed of cells, and each cell can either be empty, or occupied by exactly one vehicle Nagel (1992); Nagel and Schreckenberg (1992), see Fig. 32.1 (a). Since movement has to be from one cell to another cell, velocities have to be integer numbers between 0 and v_{max}, where the unit of velocity is [cells per time-step]. It turns out that reasonable values are Nagel and Schreckenberg (1992); Barrett et al. (1995):

- length of a box $= 1/\rho_{jam} =$ 7.5 m ($\rho_{jam} =$ density of vehicles in a jam).
- time step $=$ 1 sec
- maximum velocity $=$ 5 boxes per time step $= 5 \cdot 7.5$ m/sec $=$ 135km/h $\approx$ 85mph

For other conditions, such as higher or lower speed limits, this can be adapted.

Note that this approach implies a *coarse graining* of the spatial and temporal resolution and therefore of the velocities. A vehicle which has a speed of, say, 4 in this model stands for a vehicle which has a speed anywhere between $3.5 \cdot 7.5$ meters/sec ≈ 95 km/h (59 mph) and $4.49999 \cdot 7.5$ meters/sec ≈ 121 km/h (75 mph).

Vehicles move only in one direction. For an arbitrary configuration (velocity and position), one update of the traffic system consists of two steps: a velocity update step consisting of three consecutive rules, and a movement step according to the result of the velocity update. The whole update is performed simultaneously for all vehicles. The complete configuration at time step t is stored and the configuration at time step $t+1$ is computed from that "old" information. Computationally we calculate in time step t (with the three rules) the new velocity of each car and write this newly calculated velocity in the same site without moving the car (velocity update). After that we move all cars according to their newly calculated velocity (movement update).

1. (velocity update)

   For all particles i simultaneously, do the following:

   IF ($v_i \geq gap_i$)

   $$v_i := \begin{cases} gap_i - 1 & \text{with probability } p_{noise} \text{ if possible}^3 \\ gap_i & \text{else} \end{cases} \quad \text{(close following/braking)}$$

   ELSE IF ($v_i < v_{max}$)

   $$v_i := \begin{cases} v_i & \text{with probability } p_{noise} \\ v_i + 1 & \text{else} \end{cases} \quad \text{(acceleration)}$$

   ELSE (i.e. ($v_i = v_{max}$ AND $v_i < gap_i$)

   $$v_i := \begin{cases} v_{max} - 1 & \text{with probability } p_{noise} \\ v_{max} & \text{else} \end{cases} \quad \text{(free driving)}$$

   ENDIF

2. (movement update)

   Move all particles i to $x_i(t+1) = x_i(t) + v_i$.

The index *i* denotes the position (an integer number) of a vehicle, *v*(*i*) its current velocity, v_{max} its maximum speed, *gap*(*i*) the number of empty cells ahead, and p_{noise} is a randomization parameter.

The first velocity rule represents noisy car following or braking. If the vehicle ahead is too close, the vehicle itself attempts to adjusts its velocity such that it would, in the next time-step, reach a position just behind where the vehicle ahead is at the moment. Yet, with probability p_{noise}, the vehicle is a bit slower than this.

The second velocity rule represents noisy acceleration. Essentially, the acceleration is linear (i.e. independent from current speed), but with probability p_{noise}, no acceleration happens in the current time step (maybe as a result of switching gears etc.). Instead of an acceleration sequence of $0 \to 1 \to 2 \to 3 \to \ldots$, a possible acceleration sequence can now be $0 \to 0 \to 1 \to 2 \to 2 \to 2 \to 3 \to \ldots$.

The last velocity rule represents free driving. Instead of remaining always at the same speed, such vehicles fluctuate between v_{max} (with probability $1 - p_{noise}$) and $v_{max} - 1$ (with probability p_{noise}). Note that a vehicle which is set to $v_{max} - 1$ will go through the acceleration step next time, thus in the next time step either staying at $v_{max} - 1$ with probability p_{noise} or getting back to v_{max}. Note that the resulting average speed of a freely driving vehicle is thus $v_{max} - p_{noise}$.

In terms of a microscopic foundation, the model is composed of the following elements:

- If a vehicle does not have enough space ahead, speed is proportional to space headway, which implies constant time headway (Pipes' theory, May (1990)).

- If there is enough space ahead for the given velocity, the vehicle accelerates linearly either to maximum speed or until the space headway becomes too small for further acceleration. A more realistic acceleration would probably be proportional to $1/v$, where v is the speed. This would be computationally more burdensome; nevertheless, studies about the effect are under way. The effect on the principal traffic dynamics seem to be minimal Nagel and Paczuski (1995).

- Deceleration is instantaneous within one time step. If one wants to constrain the model to realistic deceleration values, one needs to look at the velocity of the vehicle ahead. This is again computationally more burdensome, and the precise difference when changing this element alone with respect to the traffic dynamics is unclear Krauß et al. (1997) although it is clear that throughput will go up Barrett et al. (1996).

- On top of these rules, we add a fairly large amount of random noise in the velocity decision.

Somewhat shorter, the model enforces constant time headway for close following and for braking, but acceleration is "delayed". This puts the model into a large class of dynamically similar models which use "time delayed" constant time headway, e.g. of the type $a(t) \propto V[\Delta x(t)] - v(t)$ or $v(t+\tau) \propto V[\Delta x(t)]$, where a is the acceleration, v is the vehicle's velocity, Δx is the space headway, $V(\Delta x)$ is a desired speed function, and τ is a time delay. It is certainly arguable that this does not catch all aspects of traffic; yet, all of these models are remarkably robust with respect to their traffic dynamics behavior, both in microscopic Krauß et al. (1997); Nagel (1996); Bando et al. (1995) and in fluid-dynamical Kühne and Beckschulte (1993); Kerner and Konhäuser (1994) implementations.

32.4.2 Lane changing for passing

For multi-lane traffic, the model consists of parallel single lane models with additional rules for lane changing. Here we describe the two lane model which can be modified to any kind of multi lane model. Lane changing is modeled by an additional update step, which is added before the velocity update. The new sequence of steps is presented below. Steps two and three are the same in the single lane model and they are executed separately for each lane.

1. **Lane changing decision**
2. **Velocity update**
3. **Vehicle movement**

According to this lane changing rule set the vehicles are only moving sideways during the lane changing step; forwards movement is done in the vehicle movement step. One should, though, look at the combined effect of the lane changing and vehicle movement, and then vehicles will usually have moved sideways *and* forwards. The decision to change lane is implemented as strictly parallel update, i.e. each vehicle is making its decision based upon the configuration at the beginning of the update.

- **Lane changing decision for passing**
  - **IF** neighboring position $x_o(i)$ in other lane is vacant
    - **THEN** Calculate:
      - *gap*(*i*) Gap Forward in Current Lane,

· $gap_o(i)$ Gap Forward in Other Lane,
· $gap_b(i)$ Gap Backward in Other Lane,
· **IF** ($gap(i) < v(i)$ AND $gap_o(i) > gap(i)$)
  – **THEN** $weight1 = 1$
  – **ELSE** $weight1 = 0$
· $weight2 = v(i) - gap_o(i)$
· $weight3 = v_{max} - gap_b(i)$.

* **IF** ($weight1 > weight2$) AND ($weight1 > weight3$)[4]
  · **THEN** mark vehicle for lane change[5]

The rules are working in the following way (see Fig. 32.1 (b)): First we look at the neighboring position in the target lane. If this cell is vacant, we calculate the gap forward in the current lane (gap), the gap forward in the target lane (gap_o), and the gap backward in the target lane (gap_b). With these results we calculate the $weight1$ to $weight3$ described above. Finally if the weight comparisons render true the car will change to the new lane. After executing the lane changing decision we calculate the new velocity for all cars and move them according to this velocity.

This lane changing implementation follows a usual structure Sparmann (1978); Gipps (1986):

- Reason to change lanes? (Slow car ahead? Need to make turn later (see below)?)
- If yes: Target lane empty? (Definition of "empty" depends on "urgency")
- If yes: change lanes except for stochastic noise

Lane change implementations using this framework are remarkably robust in their dynamic behavior Sparmann (1978); Gipps (1986); Rickert et al. (1996b); Nagel et al. (1998). This allows us, for example, not to look at other vehicle's velocities: The forward condition for the target lane, $gap_o \geq v$, is consistent with the condition $v \leq gap$ for the car following; the backward condition for the target lane, $gap_b \geq v_{max}$ is simply a worst case scenario which nevertheless does not perform, in the analysis of the emergent properties, any worse than a condition which depends on the velocity of the other car (compare, e.g., Wagner et al. (1997) with Wagner (1996)).

For three or more lanes, a simultaneous implementation of the lane changing decision can lead to collisions. For example, in a three-lane road two vehicles on the left and right lane could decide to go to the same spot in the middle lane. From an algorithmic point of view, this is possible because the lane changing decision is based on the configuration on time t; but it is also an entirely realistic situation.[6] To avoid collision we only allow lane changes in a certain direction in each time step:

- **IF** the time step is even

  THEN start procedure *lane changing decision* to the *left* for cars on the middle and then on the right lane

[4]Weights are used because of extensibility towards "lane changing for plan following". See below.

[5]In the current version, the lane change is actually still rejected with a probability of 0.01 even when all the rules are fulfilled. This is in order to break the following artifact or variations of it: Assume one lane is completely occupied and one is completely empty. The above rule set will result in these vehicles just changing back and forth between the lanes—the vehicles will never get smeared out across the lanes. See Ref. Rickert et al. (1996a) for more details.

[6]In a deeper sense, the problem is caused by the fact that the underlying decision making dynamics has a time scale which is smaller than the time resolution of the simulation. The simulation thus must resolve the conflict by other means Barrett (Personal communication).

- **IF** the time step is odd

  THEN start procedure *lane changing decision* to the *right* side for cars on the middle and then on the left lane

Thus, left lane changes occur only on even time steps, right lane changes occur only on odd time steps. This behavior is collision free.

32.4.3 Lane changing for plan following

Vehicles in Transims follow route plans, i.e. they know ahead of time the sequence of links they intend to follow. This means that, when they approach an intersection, they need to get into the correct lanes in order to make the intended turn. For example, a vehicle which intends, according to its route plan, to make a left turn at the next intersection needs to get into one of the lanes which actually allow a left turn.

This is achieved in Transims by supplementing the basic lane changing rules with a bias towards the intended lanes. This bias increases with increasing urgency, i.e. with decreasing distance to the intersection. Technically, this is achieved by adding another weight to the acceptance conditions for lane changing:

- **IF** ($weight1 + weight4 > weight2$) AND ($weight1 + weight4 > weight3$)

  THEN change lane

$weight4$ is calculated according to

$$weight4 = \max\left[\frac{d^* - d}{v_{max}}, 0\right] \tag{32.1}$$

for lane changes in the desired direction as long as the vehicle is not in one of the correct lanes, cf. Fig. 32.1 (c). d is the remaining distance to the intersection, d^* is a parameter; both are given in the unit of "cells". d^* is currently set to 70 cells, i.e. approx. 500 m or 1/3 of a mile, throughout the simulation. In consequence, $weight4$ increases from zero to $d^*/v_{max} = 14$ during the approach to the intersection. If $weight4 = 0$, then it does not influence lane changing decision. $weight4 = 1$ has the same effect as a slower vehicle ahead on the same lane. Further increases of $weight4$ more and more override the security criterions that the forward and the backward gap on the destination lane need to be large enough. $weight4 > v_{max}$ lets the vehicle make the lane change even if only the neighboring cell on the destination lane is free.

Once a vehicle is in one of the "correct" lanes within 70 cells (525 m) of the intersection, it is only allowed to change lanes if the target lane is also "correct". For movements that are allowed on multiple lanes through the intersection, this leads to equal usage of these lanes. This algorithm is *not* capable of leaving a single "correct" lane temporarily when encountering, say, a stopped bus on the same lane.

32.4.4 Unprotected turning movements

A necessary element of traffic simulations are unprotected turning movements. By this we mean that that for the movement the driver intends to make, some other lanes have priority. Examples are stop signs, yield signs, on-ramps, unprotected left turns.

The general modeling principle for this in Transims is based on a gap acceptance in the opposing (in Transims sometimes called "interfering") lanes, see Fig. 32.1 (d). Opposing lanes are the lanes which have priority; for example, for a stop-controlled left turn onto a major road this would be all lanes coming from the left plus the leftmost lane coming from the right. In order to accept the turn, there has to be a sufficient gap in each of these lanes.

Note that "gap divided by the velocity of the oncoming vehicle" is the oncoming vehicle's time headway, so the dynamics of this follows the Highway Capacity Manual Transportation Research Board (1994b). If one wants a time headway on an opposing lane of at least 3 seconds, then a vehicle with a velocity of 4 cells/second would have to be at least 12 cells away from the intersection.

The current Transims microsimulation uses a gap acceptance (gap between intersection and nearest car to the intersection which is approaching) of 3 times the oncoming vehicle's velocity, i.e. when the gap on each opposing lane is larger than or equal to the first vehicle on that lane, the move is accepted. For example, if the oncoming vehicle has a speed of 3, at least 9 empty cells have to be between the oncoming vehicle and the intersection. A special case is if the oncoming vehicle has the velocity zero, in which case no gap is necessary.

32.4.5 Signalized intersections

In Transims, we distinguish between signalized intersections and unsignalized intersections. In signalized intersections, the priorities are changing in time and regulated by signals. In unsignalized intersections, the priorities are fixed.

When a simulated vehicle approaches a *signalized* intersection, the algorithm first decides if, according to its current speed, it potentially wants to leave the link, i.e. its current speed (in cells per update) is larger than or equal to the remaining number of cells on the link.[7] If a vehicle wants to leave the link, the algorithm checks the "traffic control", which determines if the vehicle can leave the link. If it encounters a red light, it can *not* leave the link and no further action is taken. If it encounters a protected (green arrow) or caution (yellow) signal, the vehicle is allowed to enter the intersection. If it encounters a permitted signal (green, for example permitted left turn against oncoming traffic), the vehicle checks all opposing flows for a gap that is larger or equal to 3 times the oncoming vehicle's velocity (see Subsec. 32.4.4 above).

If the movement into the intersection is accepted, the vehicle is moved into an "intersection queue"; there is one queue for each incoming lane. This queue models vehicle behavior inside an intersection. The vehicle gets a "time stamp", before which it is not allowed to leave the intersection; this time stamp is representative of the duration of the movement through the intersection. The intersection queues have finite capacity; once they are full, no more vehicles are accepted and the vehicles start to queue up on the link. This models the finite vehicle storing capacity of an intersection.

Once a vehicle is ready to leave the intersection, it moves to the first cell on the destination link if available. The speed of the vehicle is not changed when it is in the intersection queue so it exits on the destination link in the first cell with the same velocity that it had when it entered the queue.

Note that vehicles turning against opposing traffic make their decision to accept the turn when they *enter* the intersection queue, not when they leave it. This can have the effect that a vehicle enters the intersection queue when there is no oncoming traffic, but, because of other vehicles ahead of it in the same queue, cannot make its turn immediately. Yet, since the turn was already accepted, it will be executed as soon as all vehicles ahead in the same queue have cleared the queue and a cell on the destination link is available. The turn can occur during oncoming traffic. So in some sense vehicles will go "through" each other. Yet, note that on average the result is still correct. The approach described above will not let more vehicles through the intersection than a gap acceptance calcu-

[7] Vehicles may accelerate or slow down before they actually reach the intersection. See below.

lated when *leaving* the intersection queue. The above logic was chosen for simplification purposes since unsignalized intersections (see below) do not have queues and thus *need* to make their acceptance decisions when entering the intersection.

32.4.6 Unsignalized intersections

Unsignalized intersections in Transims have no internal queues, i.e. vehicles go right through them.[8] Also, vehicles leaving an unsignalized intersection go down the destination link as far as prescribed by their velocity, not just into the first cell as in the signalized intersections. Apart from these two differences, unsignalized intersections are similar to signalized ones.

When a simulated vehicle approaches an unsignalized intersection, the algorithm first decides if, according to its current speed, it potentially wants to leave the link, i.e. its current speed (in cells per update) is larger than or equal to the remaining number of sites on the link. If a vehicle wants to leave the link, the algorithm checks the "traffic control", which determines if the vehicle can leave the link. Currently occuring traffic controls are: no control, yield, and stop.

If a "no control" is encountered, the vehicle is moved to its destination cell without any further checks. For example, if a vehicle has a velocity of 5 cells per update and 2 more cells to go on its link, then it attempts to go 3 cells into the destination link. If that cell is already reserved (either by another "reservation" or by a real vehicle), then the next closer cell is attempted, etc., until the algorithm either finds an empty cell or returns that the destination lane is full. "No control" is usually used for the major directions, i.e. for the lanes which have priority.

If a yield sign is encountered, the vehicle checks the gap on all opposing lanes. According to the same rules as above, on all opposing lanes the gap needs to be larger or equal three times the first vehicle's speed on that lane. If the movement is accepted, the destination cell is selected according to the same rules as with the "no control" case.

If it encounters a stop sign, the vehicle is brought to a stop. Only when the vehicle has a velocity of zero for at least one time step on the last cell of the link is it allowed to continue. If the result of the regular velocity update indeed accelerates the vehicle,[9] then it attempts to go through the intersection. On all opposing lanes the gap, according to the same rules as above, needs to be larger or equal to three times the first vehicle's speed on that lane. If the movement is accepted, a vehicle coming from a stop sign will always go to the first cell on the destination link (if empty) and will have a velocity of one.

32.4.7 Parking locations

In the current Transims microsimulation, vehicular trips start and end at parking locations. Each link in the microsimulation, except for freeway ramps, freeway links, and some "virtual" links such as centroid connectors, has at least one parking location. Parking locations thus represent the aggregated parking options on that link. Parking locations have rules about how vehicles enter and exit the simulation:

- Each vehicle in Transims has a complete route plan, together with a starting time. At the starting time, the vehicle is added to a queue of vehicles that want to leave the same parking location. When the vehicle is the first one in the queue, it attempts to enter the link. The acceptance logic is in spirit similar to the logic of the

[8] Again, technically the vehicles only reserve cells on the destination links. The actual move through the intersection happens later and can also be postponed if after the velocity update the vehicle actually does *not* make it to the intersection.

[9] I.e. there is a probability of $1 - p_{noise}$ that the vehicle will not accelerate in the given time step.

unsignalized intersections, i.e. vehicles check the available gap and make their decision based on that. Parking accessory logic is not the focus of the current paper, and since that logic may change in Transims in the near future and we also expect no influence on the results presented here, we omit further technical details.

- A vehicle that has reached its destination parking location according to its plan will leave the microsimulation.

32.4.8 Parallel logic

Transims is designed to run on parallel computers, such as coupled workstations, desktop multi-processors, or supercomputers. The parallelization approach used for the microsimulation is a geographical distribution, i.e. different geographical parts of the simulated area are computed on different CPUs.

The current Transims microsimulation has these boundaries always in the middle of links. This is done in order to keep the complexity of the parallel computing logic as far away as possible from the complexity of the intersection logic.

Information needs to be exchanged at the boundaries several times per update in order to keep the dynamics consistent. For example, if a vehicle changes lanes and ends up close in front of another one, that other one is probably forced to brake. Now, if the lane changing vehicle is on one CPU and the following one on another, one needs to communicate the lane change. This will be called "Update boundaries" in the following section.

32.4.9 Complete scheduling

For a complete transportation microsimulation, we need to specify when movements are accepted, and also how conflicts are resolved. For example, vehicles simultaneously attempting to change lanes into the middle lane represent such a conflict. Another conflict is two vehicles from two different links competing for the same site on the destination link.

The complete update of the current Transims microsimulation is as follows. Assume that the state at time t is the result of the last update. Let $t1$, $t2$, etc. be intermediate partial time steps.

1. Vehicles which are ready to leave intersection queues from signalized intersections reserve cells on outgoing lanes. They only attempt to reserve the first cell on the link; their velocity is the same as it was when they entered the intersection. When the cell is occupied (either by another "reservation" or by a vehicle), then the vehicle cannot leave the intersection. Note that there can be a conflict between different queues for the same destination cell. The current solution in Transims is that queues are served on a first come first served basis in some arbitrarily defined way, i.e. a queue which happens to be treated earlier in the microsimulation has a slightly higher chance of unloading its vehicles. — Result: t_1 information.

2. Vehicles change Lanes. Use information from time t_1 to calculate situation at time t_2.

3. Exit from Parking. Results in t_3 information.

4. Exchange boundary information for parallel computing.

5. Non-signalized intersections reserve sites on target lanes. Note that there can be a conflict of two incoming links competing for the same destination cell. The

current solution in Transims is that links are served on a first come first served basis, i.e. a link which happens to be treated earlier in the microsimulation has a slightly higher chance of unloading its vehicles. Note that this conflict only happens between minor links. Major links never compete for the same outgoing link except when there is a network coding error; and for the competition between major and minor links, the major link always wins because of the opposing lanes conditions.[10] Result: t_4 information.

6. Calculate speeds and do movements. If a vehicle scheduled for an intersection does not go through the intersection as a result of the velocity update, the reservation is cancelled. Vehicles which go through unsignalized intersections have p set to zero, i.e. if it turns out that the result of the velocity update indeed brings them into the intersection, they need to go to the site on the destination lane which was reserved earlier. Result: $t_5 = t + 1$ information.

7. Exchange boundary information and migrate vehicles for parallel computing.

32.5 Towards a standardized flow test suite for simulation models

In order to control the effect of driving rules, Transims provides controlled tests for traffic flow behavior. These tests are simplified situations where elements of the microsimulation can be tested in isolation. This test suite uses the standard microsimulation code in the same way it is used for full-scale regional simulations, and it also uses the same input and output facilities: The test network is currently defined via a table in an Oracle data base, in the same format as the Dallas/Fort Worth network is kept. Input of vehicles is, following individual vehicle's plans, via parking locations, the same way vehicles enter regional simulations.[11] Output is collected on certain parts of the network on a second-by-second basis, the same way it can be collected for regional microsimulations. The collected output is then post-processed to obtain the aggregated results presented in this paper.

The test cases we look at in this paper are the following (see also Fig. 32.1 (e)):

- One-lane traffic, in order to see if car following behavior generates reasonable fundamental diagrams.
- Three-lane traffic, in order to see if the addition of passing lane changing behavior still generates reasonable fundamental diagrams, and in order to look at lane usage.
- Stop sign, yield sign, and left turns against oncoming traffic, in order to see it the logic for non-signalized intersections generates acceptable flow rates.
- A signalized intersection, in order to see of we obtain reasonable flow rates, and in order to check lane changing behavior for plan following purposes.

32.5.1 Measured quantities

We look at three minute averages of the following quantities:

[10] Note that the situation slightly different when the speed of the vehicle on the major link is zero – see below.

[11] Route plans are simply necessary to be consistent with the way the simulation is normally used; for the test cases we use very few types of generic route plans (like "enter the microsimulation and keep on driving in a circle indefinitely") and replicate them with different starting times to fulfill our needs. This is not much different from departure rates.

- **Flow, Volume.** Flow q is defined as usual by:

$$q = \frac{N}{T} \qquad [vehicles/hour]$$

  N is the number of cars which pass a certain site at a time period *T*.

- **Density.** Density is in principle easily defined, $\rho = N/L$, where N is the number of vehicles on a piece of roadway of length L. Yet, given current sensor technology, this is not easy to achieve since one would need a sensor which counts, say once a second, cars on a predefined stretch of length L of the roadway. For that reason, empirical papers sometimes resort to occupancy, which is the fraction of time a given sensor has been occupied by a vehicle. Currently Transims measures density according to its original definition, i.e., once a time step, we count the number of vehicles on a stretch of roadway of $L = 5$ sites $= 5 \times 7.5$ m $= 37.5$ m.[12] We add these counts for $k = 180$ measurement events and then divide the resulting number by L and by k:

$$\rho = \frac{N}{k * L}$$

  The result can be scaled to convenient units, for example "vehicles per km".

  Note that this way of computing density averages the counts over a length of 37.5 m, which is longer than most traffic detectors. The effect of this should be systematically studied.

- **Space Mean Speed, Travel Velocity.** It is well known that one can measure velocity either analogous to our flow definition (Time Mean Speed, Spot Speed) or analogous to our density definition (space mean speed, travel velocity). Under non-stationary conditions, the measurements give different results, since, for example, the first definition never counts vehicles with velocity zero. Time mean speed is easier for field measurements; space mean speed is easier to interpret since it is equal to the travel velocity and it is also the velocity which needs to be used in the fundamental relationship between flow, density, and velocity, $q = \rho \cdot v$. Since in a simulation model both are similarly easy to measure, we measure the more meaningful travel velocity. Once a time step, we sum up the individual velocities of all vehicles on a stretch of roadway of $L = 5$ sites $= 5 \times 7.5$ m $= 37.5$ m. We add these sums for $k = 180$ measurement events and then divide the resulting number by N and by k, where N is the same number as obtained during the density measurement above:

$$v = \frac{\sum v}{k * N} \tag{32.2}$$

- **Lane usage.** Lane usage of a particular lane is the number of cars on this lane divided by the number of cars on all lanes. It can be computed as:

$$f_i = \frac{\rho_i}{\sum_{j=1}^{n} \rho_j \cdot n} , \tag{32.3}$$

  where i is the lane we look at and n is the number of lanes.

[12] The "magical" number of $L = 5$ sites is equal to the maximum velocity of $v_{max} = 5$ sites/update. This ensures that each vehicle is counted at least once.

32.5.2 Test networks

Essentially two test networks are used: a circle of 1 000 sites $=$ 0.75 km in various configurations, and a simple signalized intersection. Most of the tests are run on the circle networks. The circle can have one, two, or three lanes. In all tests, the circle is slowly loaded with traffic via a parking location at site $x = 1$ (where the unit of x is "cells"). Velocity, flow, and density are measured on $486 \leq x \leq 490$, thus generating the fundamental diagrams for one-lane, two-lane, and three-lane traffic. Since the circle gets slowly loaded, the complete fundamental diagram is generated during one run.

For testing yield signs and stop signs, an incoming lane is added on the right side of traffic at $x = 501$. The characteristics of the incoming traffic are measured by a detector on the last 5 sites of the incoming lane. The incoming lane is operated at maximum flow, i.e. with as many vehicles as possible entering. The incoming vehicles are removed at $x = 900$ via a parking accessory. The result of this measurement is typically a diagram showing the flow of incoming vehicles on the y-axis versus the flow on the circle on the x-axis.

For testing left turns against oncoming traffic, an opposing lane is added so that it ends at $x = 500$. The traffic control here is again a "yield" logic; the difference from before is that vehicles only *traverse* the opposing traffic, they do not join it.

Last, a three-lane intersection approach is used. The left lane makes a left turn, the middle lane goes straight, the right lane makes a right turn. Incoming vehicles have plans about their intended movement at the intersection and attempt to reach the corresponding lane. The intersection has signals with 1 minute green phase and 1 minute red phase. The typical output from this run is the flow of vehicles which go through the intersection, and the number of vehicles which cannot make their intended turn because they did not reach their lane.

32.5.3 Results

The results are shown in Figs. 32.2 to 32.5.

- Single lane traffic (Fig. 32.2a) has a realistic value of maximum flow ($=$ capacity), but one may argue that it is at a somewhat low density. The problem here is that we do not include slow vehicles; introducing slow vehicles into a single lane closed circle simulation just means that all fast vehicles bunch up behind them, which does not result in a very useful fundamental diagram. In terms of the "building block" philosophy, we prefer to run the single lane test with identical vehicles.

- Our lane changing rules do neither change maximum flow per lane nor the density (per lane) at maximum flow. That need not be the case, Rickert et al. (1996b). Again, the density at maximum flow seems a bit low. This changes considerably when one introduces slower vehicles: The free flow part of the curve then bends more to the right and the maximum is at higher densities Nagel et al. (1998). Also, there are measurements in Germany where traffic *with* trucks reaches maximum flow at approx. 20–22 veh/km/lane Wiedemann (1995), so without more specific data this discussion seems pointless. – We think that the curve without slow vehicles is "cleaner" and thus facilitates comparison between models; in reality, the problem is more complicated anyway.

  Also, we generate equal lane usage between the lanes, as should be expected for a symmetric lane changing model (in the absence of on-ramps).

- The flow through a traffic signal that is 50% green should be at half the value of the maximum single lane flow, i.e. at 1000 veh/hour, which is what we find (Fig. 32.4).

- The curves for traffic through stop and yield signs follows the general form of the curve of the Highway Capacity Manual Transportation Research Board (1994b). We added the HCM curves for comparison only. In general, we find that a yield sign, when there is no traffic on the major road, generates the same traffic as if there were no sign at all, which should be expected the way the simulation is set up. (It is a bit lower than for the "circle" before because the speed limit is lower here.) The stop sign generates a much lower flow in the same situation, because the explicit stop decreases capacity.

  From there, the curves for "traffic into" the major road decrease roughly linearly to zero when the flow on the major road reaches capacity. The curve for traffic across a single lane road looks similar to its "traffic into" counterpart, which is to be expected because the number of opposing lanes is one in both cases. The curve for traffic across a two lane road provides roughly half the flow of traffic across a single lane road.

  For densities above capacity on the major road, all curves bend "back on themselves". If the major road is congested, the speed there is zero, and the gap acceptance criterion "accept if $gap \geq 3 \cdot v_{oncoming}$" is always fulfilled, even for $gap = 0$. Nevertheless, for "traffic into", very little traffic makes it through the yield or the stop sign. The reason is that in Transims, vehicles on the major road that may go through the intersection "reserve" the first cell at the beginning of the next link, thus blocking this link for vehicles from the minor link even if the gap acceptance rule would allow the movement. For "traffic across", this restriction does not exist, and many vehicles make it through the intersection, probably many more than is realistic. – Note that the HCM does not provide any information in the congested regime.

32.6 Yield sign behavior

All runs for this paper were first done with an experimental code and then repeated with the Transims production code; all results shown so far were obtained from the Transims production code. The disadvantage of an experimental code is that actual implementation in the production version may still introduce changes in the results due to small discrepancies.[13] The advantage of an experimental code is that turnover (compile times, complexity of code, etc.) is much better than with a production version. We used that advantage to test many different rules. In the following, we want to present a small subset of tests.

All results presented in this section refer to the situation of a 1-lane minor street merging into a 1-lane major street, with the intersection control being a yield sign. Fig. 32.6 (a) shows what happens if the "reservation" rule from the Transims production code is no longer used. Clearly, if vehicles from the major road do reserve cells on the outgoing link only if they are actually going there, many more vehicles from the minor lane can make the turn, effectively leading to an "alternating" vehicle pattern. This may be desirable in some situations.

Figs. 32.6 (b) shows what happens when one then changes "accept when $gap \geq 3v_{oncoming}$" to "accept when $gap > 3v_{oncoming}$". This seems like a negligible difference in the rules; yet, the results are quite different in the congested regime. Whereas in the first, many vehicles are able to get into the congested major road, in the second, only few of them make it. The difference is easiest explained by looking at a vehicle of speed zero on the major road just in front of the merge point, with space for a vehicle downstream of the

[13] This explains the differences to the TRB preprint version of this paper, which contained results from the experimental code.

merge point. With the first rule, a vehicle at the yield sign will accept the move and move in front of the vehicle on the major road, in the second case, it will not. Both scenarios seem to be plausible to us; only systematic measurements can probably resolve which one is better for a simulation model. – Also note that the rule in (b) generates similar flows as the Transims production version.

Fig. 32.6 (b), (c) and (d) show the result of different speed limits (same speed limit for both streets). A high average free speed of approx. 130 km/h (≈ 80 mph, generated by $v_{max} = 5$), maybe a freeway merge, generates a flow of approx. 2000 veh/hour/lane in the incoming lane when there is no traffic on the major road (Fig. 32.6 (c)). From there, maximum incoming flow decreases continuously. Lower average free speeds of approx. 75 km/h (50 mph, Fig. 32.6 (b)) and 50 km/h (30 mph, Fig. 32.6 (d)) generate lower maximum incoming flows and are generally closer to the Highway Capacity Manual curve. Yet, it should be clear that, contrary to the HCM, the "minor" flow is also a function of the speed limit and not only of the gap acceptance (the gap acceptance is the same in all three simulations).

A last series of experiments shows the effect of different values for the gap acceptance. Figs. 32.6 (e) and (f) show "accept when $gap > v_{oncoming}$ and $gap > v_{max}$". Clearly, more vehicles are accepted, leading to a higher flow of turning vehicles as a function of the flow on the major road. Note that the flow via the yield sign is never higher than 1800 minus the flow on the major road. This reflects the fact that the major road cannot have a higher flow than 1800 veh/h/lane (free speed approx 50 mph); traffic through the yield sign can thus at most fill the major road to capacity. This explains why the acceptance of much smaller gaps do not produce a stronger difference. The situation is clearly different for unprotected turns *across* instead of *into* traffic, as can be seen for the left turns in the next section.

32.7 Comparison to Case Study Logic

The gap acceptance logic presented here and used in the March 1998 Transims microsimulation is different from the logic used in the "Dallas/Fort Worth Case Study" Beckman et al (1997); Nagel and Barrett (1997). The logic during that case study was: "Accept an unprotected movement if in all opposing lanes the gap is larger than $v_{max} = 5$." This means that at low density on the major road, more turns were accepted, whereas at high density on the major road, less turns were accepted – with the extreme case that no turns were possible against oncoming traffic of speed zero.

Fig. 32.7 compares the results for the current gap-acceptance logic and the one used in the case study for the case where the major road is a 3-lane road. Note that the results for the turns *into* other traffic are not that much different whereas the result for the turns *across* other traffic yields much higher uncongested and much lower congested flows with the case study logic. This is due to the fact that for turns *into* other traffic, there is a capacity constraint of the form that the joint flows from the major and the incoming road cannot exceed capacity of the major road, see last section. Such a constraint obviously does not exist for turns *across* the major road.

32.8 Short discussion

We presented test of what we believe are "building blocks" of microsimulation models. Further "building blocks", not included here, are probably freeway ramps with merge lanes, and freeway weaving sections. We plan to include these tests into future versions.

As pointed out earlier, we believe that "clean" real world measurements of the "building block" situations are hard to obtain. Thus, one may consider them primarily useful for comparing simulations with each other and with theory; nevertheless, we think that one can judge from the results at least if the simulation is "in the right ballpark". It would certainly be desirable in the future to also have test suites for more complex situations. – For the same reason, we did not make any attempt to get "better" results than the ones presented here: we know that the results change in more complex scenarios, and it is therefore unclear if a change "to the better" in the test cases may not be a change "to the worse" with respect to reality.

Also, we would shortly like to point out again that "verification" of simulation models, i.e. the question if an actual code corresponds to a (possibly incomplete) specification in a paper, is in practice a difficult question. An alternative approach would be to try to find a suite that decides if we are macroscopically convincing without the need to go through testing the rules on an individual scale. Arguing about the microscopic rules could then be left to a small group of specialists; the end user could just look at the test suite results and judge in a matter of minutes if the simulation has faults that would seriously affect the analysis of their problem.

Last, all these problems imply to us that one should expect that simulation models will undergo continuous improvements, and it seems more realistic to us to expect "test suites" to be run at regular intervals instead of expecting that parts of simulation models can be validated and calibrated "once and for all" at certain stages and then never be touched again. In consequence, we would like to shift the argument from a discussion whether a model is "correct or not" to the discussion about which tests should be run to enable the user to make that decision, and how these tests can be made comparable between different simulation models.

32.9 Summary and conclusion

In transportation simulation models for larger scale questions such as planning, the flow characteristics of the traffic dynamics are in some sense more important than the microscopic driving dynamics of the vehicles itself. This becomes especially true since a "complete" representation of human driving is impossible anyway, both due to knowledge constraints and due to computational constraints. Yet, calibrating a traffic simulation model against all types of desired behavior (for example against all HCM curves and values mentioned in this paper) seems a hopeless task given the high degrees of freedom.

Transims thus attempts to generate plausible emergent macroscopic behavior from *simplified* microscopic rules. This paper described the more important aspects of these rules as currently implemented or under implementation in TransimsḂefore we implement rules in the Transims production version, we usually try to run systematic studies with more experimental versions. The results of the traffic flow behavior from that study were presented. Also, we showed the effects of some changes in the rules for the example of a yield sign. Finally, some comparisons were made between the logic currently under implementation and the logic used for the Dallas/Fort Worth case study.

One problem with microscopic approaches is that, in spite of all diligence, subtle differences between design and actual implementation can make a significant difference in the emergent outcome. For that reason, this paper should also be seen as an argument for a standardized traffic flow test suite for simulation models. We propose that simulation models, when used for studies, should first run these tests to demonstrate the dynamics of their emergent macroscopic flow behavior. We think that the combination of results presented in Figs. 32.2 to 32.5 are a good test set, although extensions may be necessary

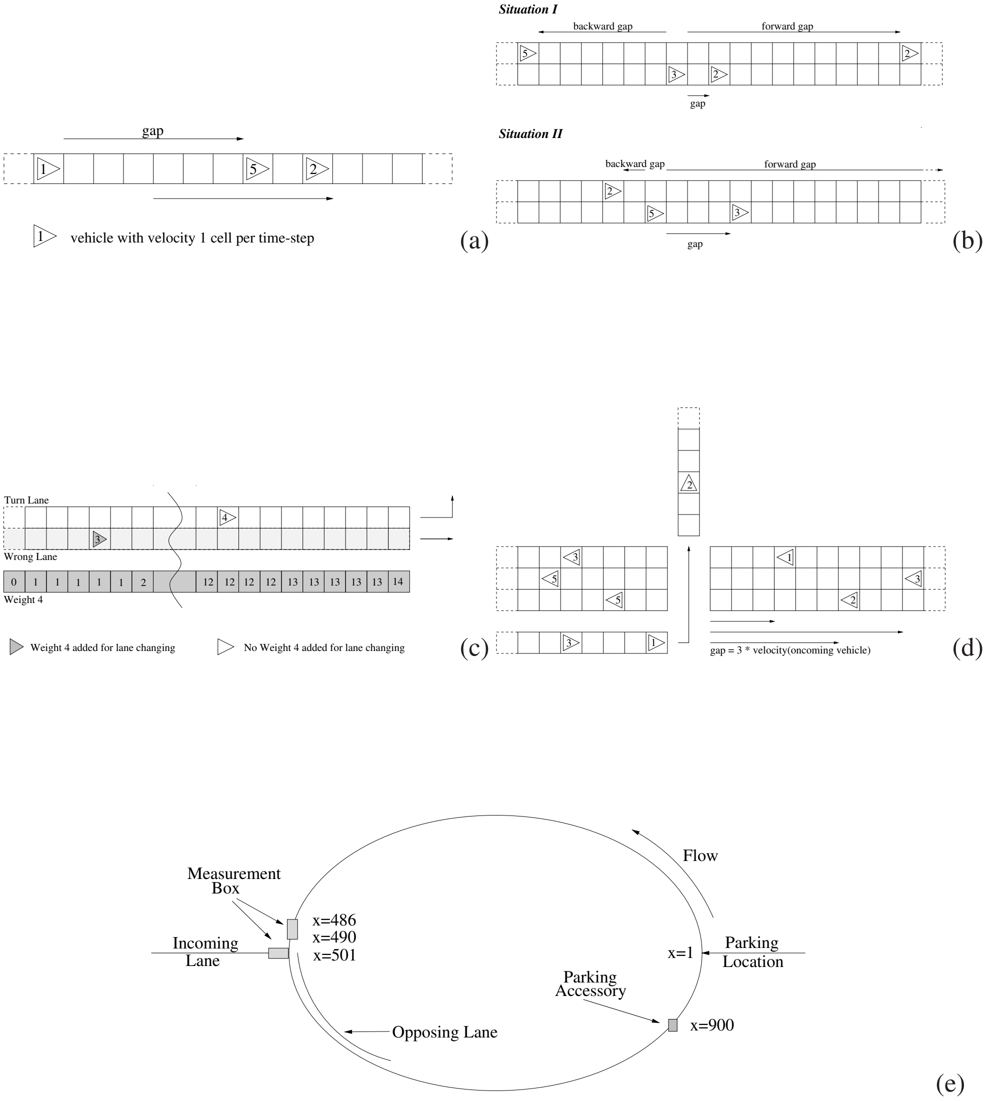


Figure 32.1: *(a)* Definition of gap and examples for one-lane update rules. Traffic is moving to the right. The leftmost vehicle accelerates to velocity 2 with probability 0.8 and stays at velocity 1 with probability 0.2. The middle vehicle slows down to velocity 1 with probability 0.8 and to velocity 0 with probability 0.2. The right most vehicle accelerates to velocity 3 with probability 0.8 and stays at velocity 2 with probability 0.2. Velocities are in "cells per time step". All vehicles are moved according to their velocities at a later phase of the update. *(b)* Illustration of lane changing rules. Traffic is moving to the right; only lane changes to the left are considered. Situation I: The leftmost vehicle on the bottom lane will change to the left because (i) the forward gap on its own lane, 1, is smaller than its velocity, 3; (ii) the forward gap in the other lane, 10, is larger than the gap on its own lane, 1; (iii) the forward gap in the target lane is large enough: $weight2 = v - gap_o = 3 - 10 = -7 < 1 = weight1$; (iv) the backward gap is large enough: $weight3 = v_{max} - gap_b = 5 - 6 = -1 < 1 = weight1$. Situation II: The second vehicle from the right on the right lane will not accept a lane change because the gap backwards on the target lane is not sufficient. *(c)* Value of $weight4$ when in wrong lane during the approach to the intersection. *(d)* Example of a left turn against oncoming traffic. The turn is accepted because on all three oncoming lanes, the gap is larger or equal to three times the first oncoming vehicle's velocity. *(e)* Test networks.

in the future (e.g. merge lanes, weaving, etc.). We will attempt to provide future Transims results also with updated versions of the results of the traffic flow tests.

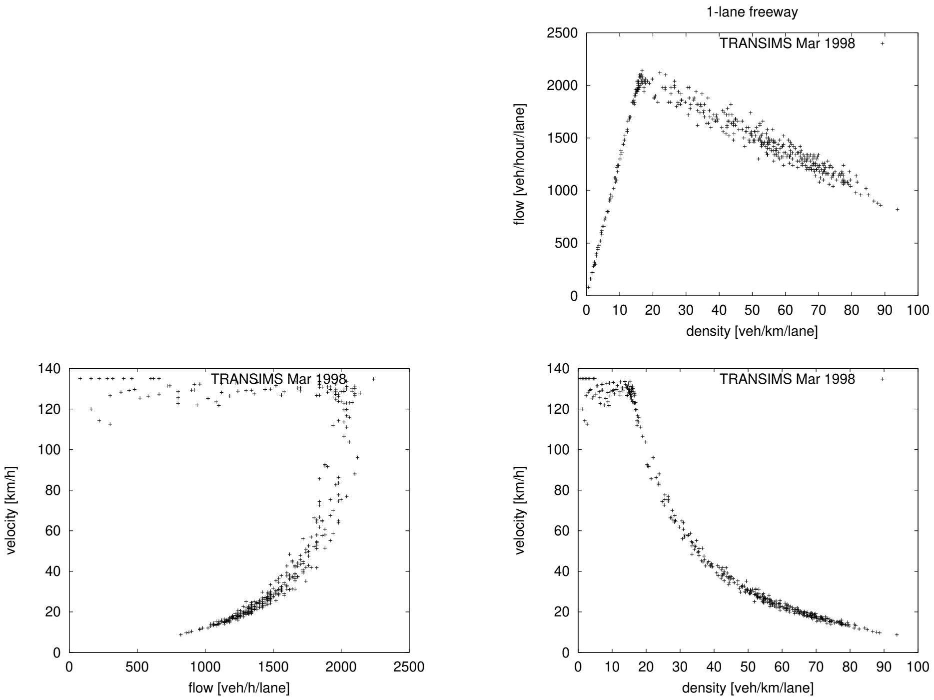


Figure 32.2: One-lane traffic: Flow vs. density, travel velocity vs. flow, and travel velocity vs. density.

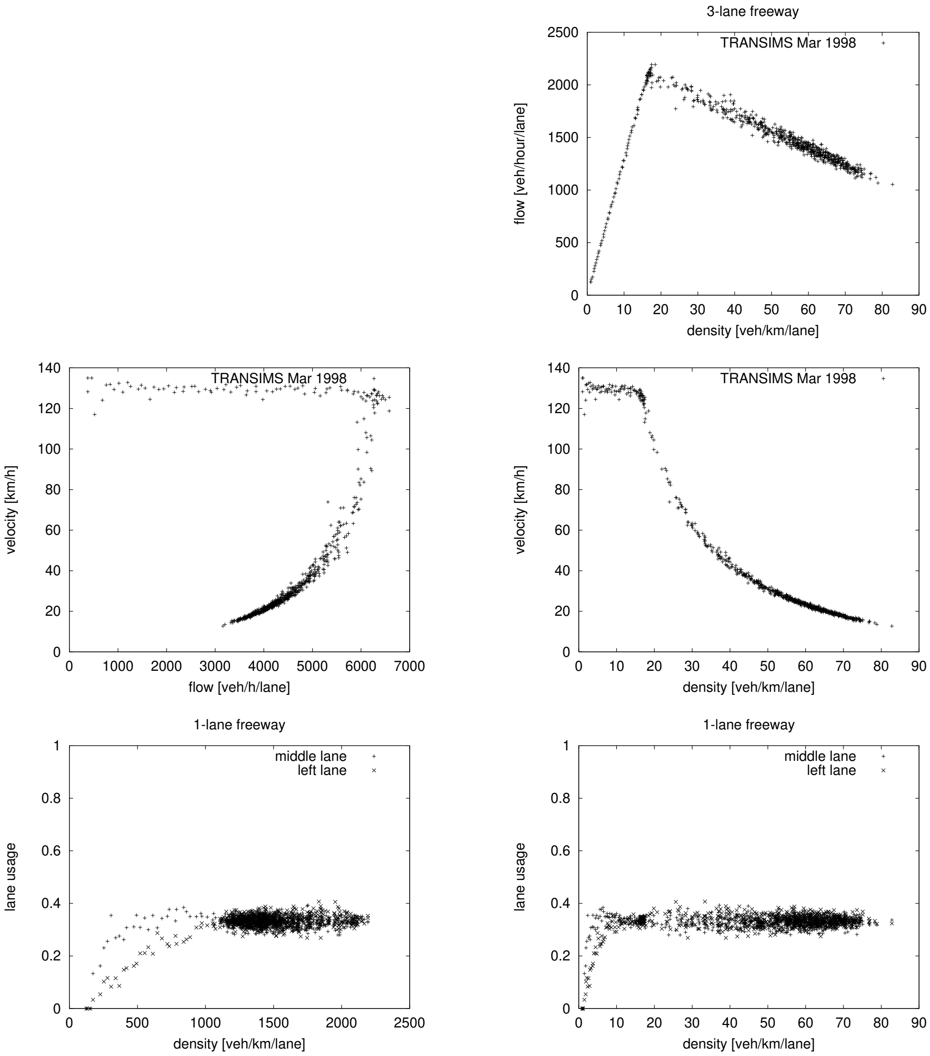


Figure 32.3: Three-lane circle: Flow vs. density, travel velocity vs. flow, travel velocity vs. density, lane usage vs. flow, and land usage vs. density. The asymmetry in the lane usage at low densities is due to the fact that the parking locations start filling in vehicles on the right lane, and they only move to the left when traffic on the right lane becomes dense.

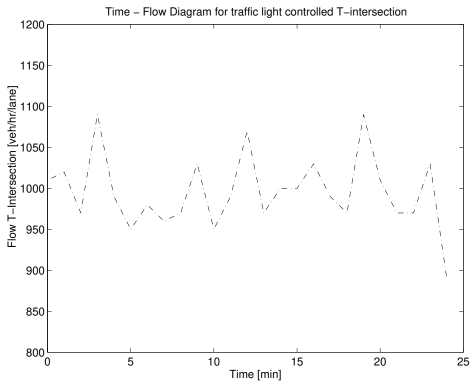


Figure 32.4: Number of vehicles going through the intersection and number of vehicles "off plan" ($= 0$) per green phase, re-scaled to hourly flow rates per lane.

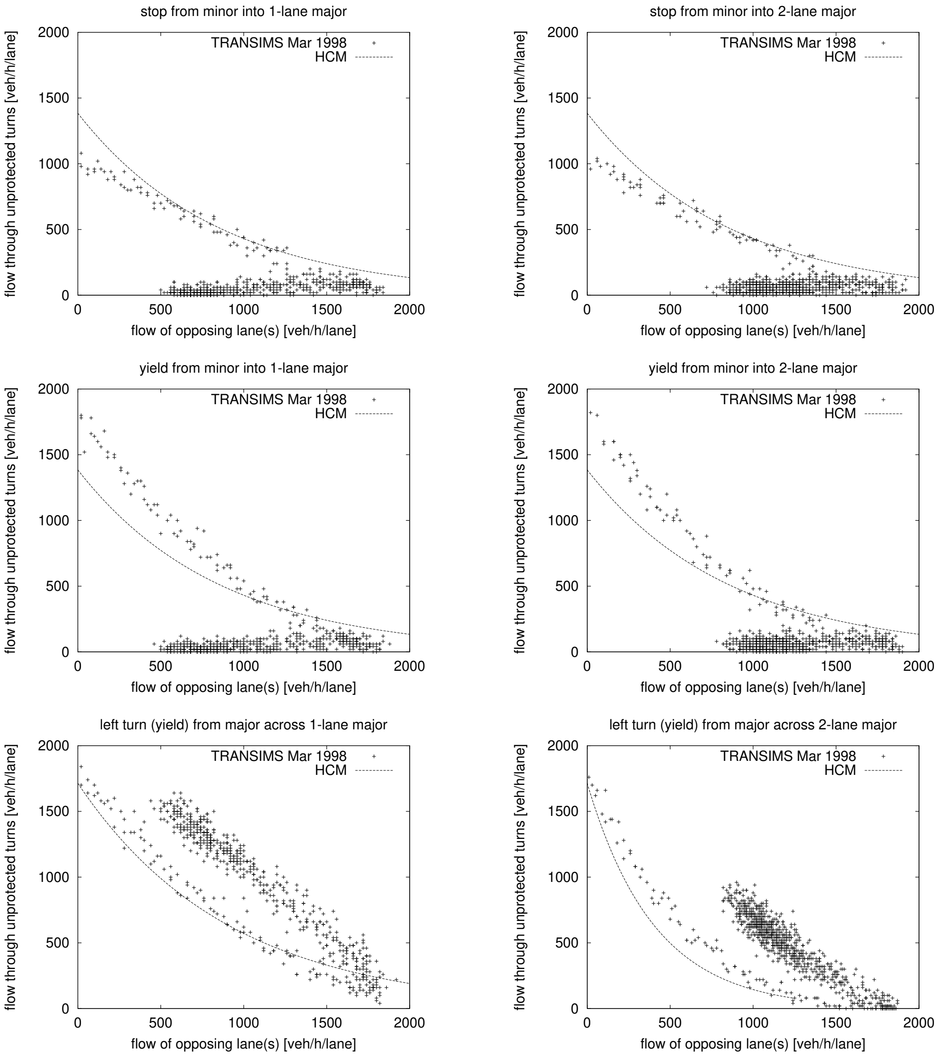


Figure 32.5: Flow through stop sign, yield sign, and unprotected left turn. Left column: Major road ("circle") has one lane. Right column: Major road ("circle") has two lanes. Solid line: Highway Capacity Manual Transportation Research Board (1994b). $v_{max} = 3$, gap acceptance rule is "accept if $gap \geq 3 \cdot v_{oncoming}$, and if first site on target lane available". Note that for "left turn across two lanes" (bottom right) the opposing volume is the sum of both lanes, i.e. twice the value shown on the x-axis.

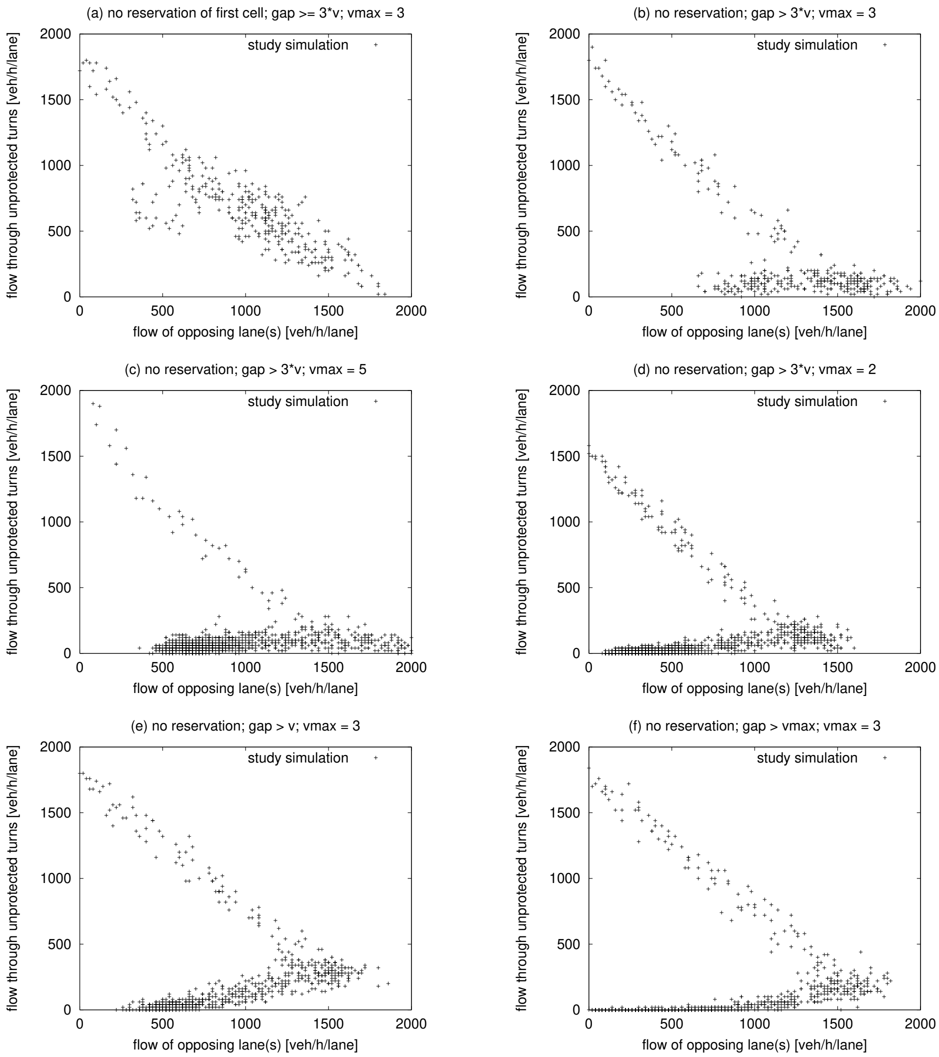


Figure 32.6: Comparison between different rules for the case of a 1-lane minor road controlled by a yield sign merging into a 1-lane major road. *(a)* Same as Fig. 32.5 (i.e. $v_{max} = 3$ and "accept if $gap \geq 3 \cdot v_{oncoming}$"), except that traffic on major road does not reserve the first cell on the outgoing link, thus giving traffic from the yield sign more opportunities. Note that this seemingly small difference has big consequences in the congested regime. *(b)* Same as (a) except that acceptance rule now "accept if $gap > 3 \cdot v_{oncoming}$". *(c)* Same as (b) except that $v_{max} = 5$. *(d)* Same as (b) except that $v_{max} = 2$. *(e)* Same as (b) except that acceptance rule now "accept if $gap > v_{oncoming}$. *(f)* Same as (b) except that acceptance rule now "accept if $gap > v_{max}$".

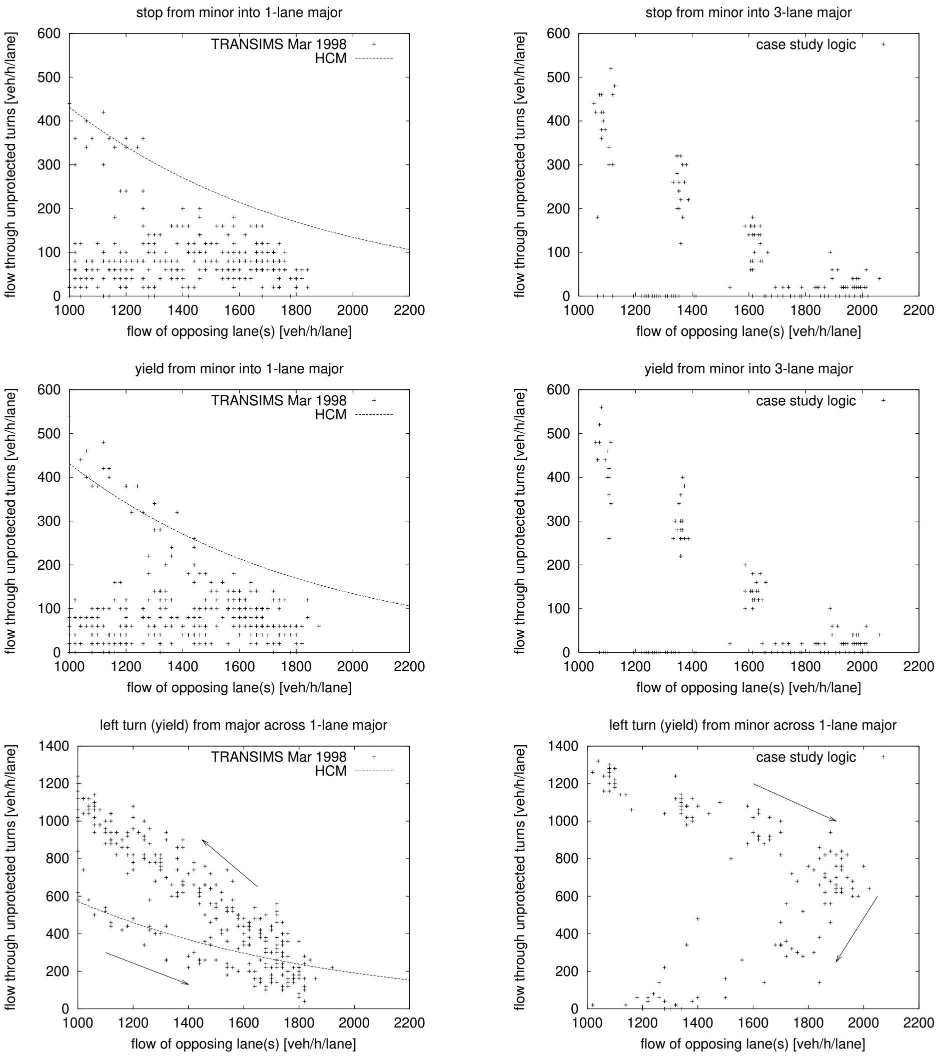


Figure 32.7: Comparison between the March 1998 Transims microsimulation gap acceptance logic and the one used in the case study. Flow through stop sign, yield sign, and unprotected left turn into/across traffic on major road. Left column: March 1998 Transims microsimulation. Right column: case study Transims microsimulation. The arrows in the left turn case indicate the direction of increasing congestion. – The results are not strictly comparable because (i) the simulations in the right column were run with a maximum speed of $v_{max} = 5$ cells/update (135 km/h) vs. $v_{max} = 3$ cells/update (81 km/h) in the left column (mostly noticeable in the lower maximum flow on the major road); and (ii) the stop and yield cases on the right describe flow into a 3-lane road vs. flow into a 1-lane raod in the left column. Note that the results for the turns *into* other traffic ("stop" and "yield") are not that much different between the two whereas the result for the turns *across* other traffic ("left turn") leads to much higher flows in the uncongested and lower flow in the congested regime with the case study logic.

Chapter 33

Intersection test suite

[[where should this go??]]

In order to systematically test this intersection logic, an intersection test suite was implemented. This test suite goes through several different intersection layouts and tests them one by one if the dynamics behaves according to the specifications. The results typically look like as shown in Fig. ??. In this particular example, one link with 500veh/sec and one link with 2000veh/sec merge into a link with a capacity of 500veh/sec. The curves are, for different algorithms, time-dependent accumulative vehicle numbers for the two incoming links. In this case, one sees that until approx time-step 3400, both links discharge at rates 400 and 100veh/sec, respectively. After that time, the first link is empty, and the second link now discharges at 500veh/sec. Not all algorithms are similarly faithful in generating the desired dynamics; the thick black lines denote results from the algorithm that got finally implemented. For further details, see Burriad (2002).

[[there is in fact a 3rd case, see daganzo network cell transmission: outgoing links and ONE incoming link congested, other incoming link not congested. Do we catch that? Do we have to?]]

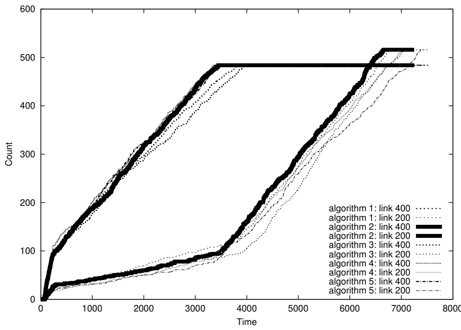


Figure 33.1: Test suite results for intersection dynamics. The curves show the number of discharging vehicles from two incoming links as explained in section 18.3.

Chapter 34

Routing

[[ben-akiva??]]

Chapter 35

A Dallas case – do I want this??

Chapter 36

A Portland/Oregon case

36.1 Introduction

Several groups are developing simulations which can microscopically simulate whole metropolitan areas in faster than real time (e.g. DYNAMIT, 2000; MITSIM, 2000; Mahmassani et al, 1995 (DYNASMART); Rickert, 1998 (PAMINA); Gawron, 1998 (LEGO); Rakha and Van Aerde, 1996 (INTEGRATION); Esser, 1998 (OLSIM)). By "microscopic" we mean that each traveller is individually resolved. Thus, if one can generate detailed travel plans for each individual, these simulations can execute these plans, while recording for example where conflicts in the form of congestion delay the plans.

In consequence, it is only a question of time until it will be easy to couple such models with models of travel demand generation, as has been demanded for many years (e.g. Axhausen, 1990). Such a coupling will probably include a modal-choice-and-routing module ("router"), and it will do systematic feedback iterations between all the modules. That is, the results of the micro-simulation will be fed back into the router again and again until some relaxation with respect to route choice is obtained, and then the result will be fed back into the activities generation module, which will generate new activities which now take into account the slower speeds in the network caused by congestion.

In this paper, an early implementation of such a computational feedback of the microsimulation into the activities module is demonstrated. In fact, practitioners have often done some version of such a feedback, by adjusting origin-destination matrices in order to move the volume counts of the assignment model closer to reality. There are also computational procedures with respect to assignment models (e.g. Metaxatos et al, 1995). What will be done here is use such a computational procedure in connection with an explicit traffic microsimulation. We will however simplify in several ways: Cars will be used as the only mode, travel from home to work will be the only demand, and the traffic micro-simulation is rather simplified. The simulation will be iteratively adjusted towards the census trip time distribution. This is an early step, and we expect much progress in the near future. In particular, we expect that transportation microsimulation, where each traveller is individually resolved, will lend itself much better to integration with activity-based demand generation than the aggregating technique of traditional assignment does. Although the focus of our work was the compuration integration of dynamic traffic assignment with demand generation, we will compare our results with existing volume counts in the Portland/Oregon area.

The structure of the paper is as follows: In Sec. 36.2, the problem is stated, followed by a description of our approach with respect to demand generation and feedback (Sec. 36.3). After a discussion of related work (Sec. 36.4), the paper moves on to our actual study

(Sec. 36.5) and its results (Sec. 36.6). The paper is concluded by a discussion and a summary.

36.2 Problem statement

In general, we want to generate "realistic traffic" via computer simulation. Thus, our ultimate research goal is to have a model which, when applied to today's situation, will yield today's traffic, and when applied to a hypothetical scenario, will yield a meaningful prediction. In our actual implementations, however, we (as everybody else) make simplifications. We are, however, not interested in optimal solutions of the simplified problems; our interest is how close to reality we can get with our simplified models and computational procedures.

We envisage that such a realistic computer simulation will be a combination of population generation, activities generation, routes assignment, and traffic micro-simulation, coupled via feedback iterations. So what is done in the following is to pick (simple) versions of these modules, embed them into feedback iterations, and try this on real world input data. The research question was twofold: (1) What are the computational issues? (2) How close to reality (or not) does one get with simple assumptions?

The question of the necessary degree of realism in each of these modules is an open problem which will need further research. That question is not treated in this paper. We do not claim that the degree of realism (or not) chosen in any of the modules used for our investigation is the correct degree of realism in order to obtain meaningful results. In particular, we expect that more sophisticated demand generation techniques (e.g. Bowman, 1998; Doherty and Axhausen, 1998; Arentze et al, 1998) will lead to more realistic results. We do expect, however, that a systematic inclusion of transportation network impedance, as demonstrated in our study, will contribute to better and more robust models.

The problem for this paper is how to assign workplace locations to workers via using computer simulation. It is known from data where people live, and it is also known where they work, but one has to match these two sets of data. The problem is similar to the trip distribution step in the four step process. In the work described here, this is done via some strongly simplified assumptions. One of these simplifications is to only look at traffic resulting from people driving from home to work. By this one neglects, for example: delivery trucks, people returning from night shifts, travelers using alternative modes of transportation, etc. There is also much more complexity in the afternoon peak than in the morning peak. Again, our investigation is a demonstration of a computational procedure, not an attempt to obtain the most possible realistic results for a certain field problem.

Having said that, let us describe our scenario. Our scenario area is Portland in Oregon. Our input data are: (a) a description of the Portland transportation network; (b) a synthetic population based on Portland demographic data; (c) a list of workplaces including location and size; (d) the distribution $N_{cns}(T)$ of actually encountered trip times T from home to work by the Portland population; and (e) a distribution of starting times. The problem for this study was to match workers (who have home locations) and workplaces such that the resulting traffic yields trip times which, when aggregated, match the census trip times.[1]

[1] Since the whole travel of each traveller in our simulation consists of exactly one trip, "trip time" and "travel time" will be used synonymously.

36.3 Our approach

The approach that is maybe closest to our work are the discrete choice models (Ben-Akiva and Lerman, 1985). As is well known, in that approach the utility V_i of an alternative i is assumed to have a systematic component U_i and a random component η_i. Under certain assumptions for the random component this implies that the probability p_i (called choice function) to select alternative i is

$$p_i = \exp(\beta U_i) / \sum_k \exp(\beta U_k) \; . \tag{36.1}$$

p_i could for example represent the probability to accept a workplace that is i seconds away. If i is indeed taken as time, then U_i is negative, and it follows an inverse S-shaped curve which starts at zero, decreases slowly for small times, decreases faster for medium times, and decreases again slowly for large times (Bowman, 1998). By this approach, our above location choice problem would be solved by weighting each given workplace according to time-distance i by p_i and then making a random draw in these probabilities. Clearly, for the discrete choice approach one needs to know the function βU_i.

In this paper, the "psychological" function βU_i is obtained from "observed" trip time distributions, using new methods of micro-simulating large geographical regions. The core idea is that an observed trip time distribution $N_{tr}(t)$ can be decomposed into an accessibility part $N_{acs}(t)$ and an acceptance ($=$ choice) function $f_{ch}(t)$

$$N_{tr}(t) = N_{acs}(t) \, f_{ch}(t) \; . \tag{36.2}$$

$N_{acs}(t)$ is the number of workplaces at time-distance t; $f_{ch}(t)$ is proportional to the probability that a prospective worker will accept this trip time. Thus, apart from normalization f_{ch} is the same as the choice function in discrete choice theory. Our decomposition allows to separate the network specific accessibility distribution $N_{acs}(t)$ from the "psychological" trip time acceptance function. In principle, $f_{ch}(t)$ as found via our relaxation method should be the same as when obtained via an estimation of a survey when suitably averaged over the whole population.

Given a micro-simulation of traffic, $N_{acs}(t)$ can be derived from the simulation result. For a given home location (and a given assumed starting time), one can build a tree of time-dependent shortest paths, and every time one encounters a workplace at time-diestance t, one adds that to the count for trip time t. The challenge is that this result depends on the traffic: Given the same *geographic* distribution of workplaces, these are farther away in terms of trip time when the network is congested than when it is empty. That is, given the function $f_{ch}(t)$, one can obtain the function $N_{acs}(t)$ via micro-simulation, i.e. $N_{acs}(t) = G[f_{ch}(.)](t)$, where G is the micro-simulation which can be seen as a functional operating on the whole function $f_{ch}(.)$. The problem then is to find the macroscopic (i.e., averaged over all trips) function $f_{ch}(.)$ self-consistently such that, for all travel times t,

$$N_{tr}(t) = G[f_{ch}(.)](t) \, f_{ch}(t). \tag{36.3}$$

For this, a relaxation technique is used. It starts with a guess for $f_{ch}(t)$ and from there generates $N_{acs}(t) = G[f_{ch}](t)$ via simulation. A new guess for $f_{ch}(t)$ is then obtained via

$$f_{ch}^{(n+1)}(t) = N_{tr}(t) / N_{acs}^{(n)}(t) \; . \tag{36.4}$$

A fraction f_{act} of all travelers will do their workplace selection again, using the new $f_{ch}^{(n+1)}$. $G[.]$ is generated again via micro-simulation, and this is done over and over again until a sufficiently self-consistent solution for $f_{ch}(t)$ is found.

Real census data is used for $N_{tr}(t)$ (see "census-100"-curve in Fig. 36.3; from now on denoted as $N_{cns}(t)$). People usually give their trip times in minute-bins as the highest resolution. Since our simulation is driven by one-second time steps we need to smooth the data in order to get a continuous function instead of the minute-histogram. Many possibilities for smoothing exist; one of them is the beta-distribution approach in Wagner and Nagel (1999). Here, we encountered problems with that particular fit for small trip times: Since that fit grows out of zero very quickly, the division N_{tr}/N_{acs} had a tendency to result in unrealistically large values for very small trip times. We therefore used a piecewise linear fit with the following properties: (i) For trip time zero, it starts at zero. (ii) At trip times 2.5 min, 7.5 min, 12.5 min, etc. every five minutes, the area under the fitted function corresponds to the number of trips shorter than this time according to the census data.

Obtaining $G[f_{ch}]$ itself via simulation is by no means trivial. It is now possible to micro-simulate large metropolitan regions in faster than real time, where "micro"-simulation means that each traveler is represented individually. The model used here is a simple queuing type traffic flow model described in Simon and Nagel (1999). However, even if one knows the origins (home locations) and destinations (workplaces), one still needs to find the routes that each individual takes. This "route assignment" is typically done via another iterative relaxation, where, with location choice fixed, each individual attempts to find faster routes to work. Rickert (1998) and Nagel and Barrett (1997) give more detailed information about the route-relaxation procedure; see also Fig. 36.1 and its explanation later in the text.

Once $f_{ch}^{(n+1)}(t) = N_{cns}(t)/N_{acs}^{(n)}(t)$ is given, the workplace assignment procedure works as follows: The workers are assigned in random order. For each employee the time distances t for all possible household/workplace pairs *[hw]* are calculated, while the home location h is fixed and taken directly from the household data for each employee. Let t_{hw} be the resulting trip time for one particular *[hw]* and $n_{wo}(w)$ the number of working opportunities at workplace w. Then, an employee in household h is assigned to a working opportunity at place w with probability

$$p_{hw} \propto n_{wo}(w) f_{ch}(t_{hw}). \tag{36.5}$$

In addition to work location, home-to-work activity information also includes the times when employees start their trip to work. These are directly taken from the household data.

The complete approach works as follows:

(1) Synthetic population generation: First a synthetic population was generated based on demographic data (Beckman et al, 1996). The population data comprises microscopic information on each individual in the study area like home location, age, income, and family status.

(2) Compute the acceptance function $f_{ch}(T)$. This is done as follows:

(2.1) For each worker i, compute the fastest path tree from his/her home location. Compute the resulting workplace distribution $N_{wp}(i,T)$ as a function of trip time T.[2]

(2.2) Average over all these workplace distributions, i.e.

$$N_{wp}(T) := \langle N_{wp}(i,T)\rangle_i := (1/N) \sum_i N_{wp}(i,T) \,, \tag{36.6}$$

where N is the number of workers, which is by definition also equal to the number of workplaces. $N_{wp}(T)$ is thus equivalent to our earlier $N_{acs}(T)$.

[2] In contrast to the routing module, no time-dependence was used here although future implementations should do so.

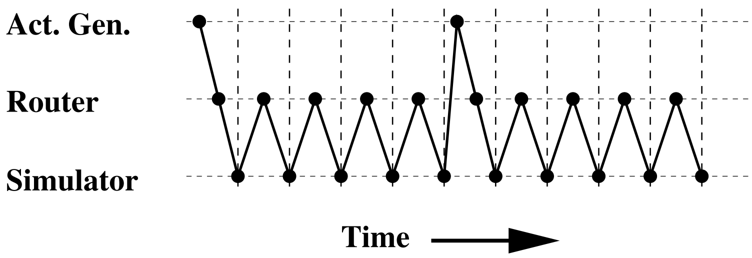


Figure 36.1: Iterative Activity Re-Assignment: Schematic subsequent application of activity generator, router, and traffic simulator.

(2.3) Compute the resulting average choice function via

$$f_{ch}(T) \propto N_{cns}(T) \,/\, N_{wp}(T) \; . \tag{36.7}$$

In addition, a normalization constant needs to be computed such that

$$\sum_T f_{ch}(T) = 1 \; . \tag{36.8}$$

(3) Assign workplaces. For each worker i do:

(3.1) Compute the congestion-dependent fastest path tree for the worker's home location.

(3.2) As a result, one has for each workplace the expected trip time T. Counting all workplaces at trip time T results in the individual accessibility distribution $N_{acs}(i, T)$.

(3.3) Randomly draw a desired trip time T^* from the distribution $N_{acs}(i, T)\, f_{ch}(T)$.

(3.4) Randomly select one of the workplaces which corresponds to T^*. (There has to be at least one because of (3.1).)

(4) Route assignment: Once people are assigned to workplaces, the simulation is run several times (5 times for the simulation runs presented in the paper) while people are allowed to change their routes (fastest routes under the traffic conditions from the last iteration) as their workplaces remain unchanged.

(5) Then, people are reassigned to workplaces, based on the traffic conditions from the last route iteration. That is, go back to (2).

This sequence, workplace reassignment followed by several re-routing runs, is repeated until the macroscopic traffic patterns remain constant (within random fluctuations) in consecutive simulation runs. For this, one looks at the sum of all people's trip times in the simulation. The simulation is considered relaxed when this overall trip time has leveled out.

Running this on a 250 MHz SUN UltraSparc architecture takes less than one hour computational time for one iteration including activity generation, route planning, and running the traffic simulator. The 70 iterations necessary for each series thus take about 4 days of continuous computing time on a single CPU.

36.4 Related work

The topic of this paper is a computational procedure of how to systematically feed back the results of a dynamic traffic assignment (DTA) to demand generation. In principle, any route assignment could be used instead of ours. However, since our work are steps towards a completely microscopic simulation approach, we are primarily interested in simulation-based route assignment and network loading. For this, one needs traffic flow simulations where one is able to follow each vehicle individually. Some simulations which fulfill this requirement besides the queue simulation used in the paper are: PAMINA (Rickert, 1998); the Transims main micro-simulation (Transims, 1992); LEGO (Gawron, 1996); INTEGRATION (Rakha and Van Aerde, 1996); DYNASMART (Mahmassani et al, 1995); PARAMICS (1996); MITSIM (Yang, 1997); DYNAMIT (2000); DYNEMO (Schwertfeger, 1987) or VISSIM (2000). Out of these, probably only LEGO, DYNASMART, DYNEMO, and DYNAMIT are fast enough to run iteration series such as ours on a single CPU. Within these four, LEGO is based on a queue model very similar to ours, while the other three use macroscopic equations for the movement of the vehicles.

In terms of re-routing during the route iterations, we use a standard time-dependent fastest path Dijkstra (see, e.g., Jacob et al, in press) based on 15-min link trip time averages. However, for this paper only a fraction of the population is re-planned. A widely used alternative is to re-plan 100% of the population in each iteration but to use a discrete choice approach approach to spread travelers across different routes (Cascetta and Papola, 1998; Bottom, 2000). Besides different theoretical properties, these approaches also have different computing complexities. The time complexity of our approach for the routing is $O(f\, N\, E \log\, K)$, where N is the number of travelers, f is the re-planning fraction (usually 10% in this paper), and $E \log\, K$ is the complexity of the Dijkstra algorithm where E is the number of edges and K the number of nodes. Note that this is independent of the time resolution. The approaches which re-plan everybody usually exploit the fact that, for any given starting location, one obtains the complete shortest path calculation for *all* destinations with the same worst case complexity as the calculation for just one destination. One thus obtains $O(F(\Delta T)\, M\, E \log\, K)$, where M is the number of possible starting points (traditionally zones) and $F(\Delta T)$ is some function that increases with increasing time resolution (decreasing ΔT) (Chabini, 1998). Since in our work each link is a potential starting point, this translates into $O(F(\Delta T)\, E^2 \log\, K)$. In this paper, where $E \approx 20k$, $N \approx 500\,000$, and $f = 0.1$, the two approaches are about equivalent. For street networks with higher resolution, E grows while N remains constant, making our approach grow more slowly in time complexity.

Also the workplace assignment is an old problem. An example of such a matching is the classic “Hitchcock” solution (Sheffi, 1985), where the workplace assignment is done in such a way that the overall sum of all trip times is minimized. This clearly results in much shorter trips than in reality. Axhausen (1990) suggests to couple demand generation, route assignment, and traffic simulation, although he puts more emphasis on on-trip learning than in the implementation presented here. Several groups such as the groups of Ben-Akiva or Mahmassani are actively working on this as extensions of their ITS projects. We are not aware of any results of these attempts yet. There are also earlier versions of the work presented in this paper (Wagner and Nagel, 1999, Esser and Nagel, 1999).

36.5 Experimental setup and simulation results

The study described in this paper was carried out as part of the Transims project (Transims, 1992), which was at that time aimed at simulating the whole city of Portland microscopically (i.e., with resolution down to single individuals) under consideration of activity generation, modal choice and route planning, and transportation dynamics. The simulations described in this paper were run on a road network consisting of 8,564 nodes and 20,024 links representing a subset of the real network.

Traffic counts for validation are available for 495 links comprising flow data for the morning peak from 7:15am to 8:15am. Data are available for the years 1992 and 1994.

Data for 1992 is used for those links for which no 1994 data are available (68 links); for all other links, the counts of 1994 are used.

The data were collected using pneumatic road tubes and averaged over two or three weekdays; mostly on Tuesdays, Wednesdays, and Thursdays outside of holiday periods and while school was in session. The counts are not seasonally adjusted. Axle adjustment factors are applied to account for trucks, which are not explicitely counted. The accuracy of the counts is considered to be $80 - 85\%$ (Bill Stein, Portland Metro, personal communication).

Another set of data available are the results of assignment runs by Portland Metro. These runs use their own demand generation, and the EMME/2 assignment algorithm (Babin, 1982). Note that "EMME/2" results in this paper will refer to results of that particular study by Portland Metro including its demand generation.

One problem with our census based assignment approach is that trip times are overestimated for at least two reasons:

(1) First, when people are asked for the time they spend for their trip to work they usually report the total door to door time including the time to get to the car or park the car. On top of that, people tend to overestimate the time they spend driving especially in stop-and-go traffic (K. Lawton, personal communication).

(2) Second, the road network used for our simulation does not cover most minor streets. That means the time people spend on these roads should be taken out of the distribution.

The amounts of those times can however not be estimated without further information. To get an idea whether a trip time distribution which is shifted to lower trip times yields more realistic results, two different workplace assignment iterations were done: One with the original census distribution, and another with all desired travel times reduced to 80% of the original value. In the following we refer to these runs as run sim-100 and sim-80, respectively.

In Fig. 36.2 the total trip time is plotted for both series, sim-100 and sim-80. Each simulation run refers to running the queue simulation for the morning (from 4am till 12pm). After every 5 iterations in which people are rerouted only, people are assigned to new workplaces. This can be seen as a sudden, normally upward jump of the total trip time in the plot. The reason for the jump is that it takes some reroute iterations to adjust the routes to the changes in the trip demand pattern. We ran 20 route iterations after the last workplace assignment to make sure that the routes are actually relaxed.

As expected, the total trip times are lower for sim-80 (Fig. 36.2). Yet, it is striking that a decrease in desired trip times by 20% results in actual trip times which are about 50% lower. The reason will be explained in the next paragraph.

By looking at the trip time distributions in the simulation (Fig. 36.3), it can be seen that the resulting distribution for sim-80 is closer to the corresponding census distribution than it is for sim-100. Even after assignment and route relaxation, there are still a lot of unrealistically high trip times for sim-100. This results from the fact that the overall traffic demand is more than the network can carry, leading to a lot of congestion. It is well known that large fluctuations occur when transportation systems are operated with demands that exceed capacities (Kelly, 1997; Nagel and Rasmussen, 1994). Actually, detailed investigation shows that in each simulation run different people account for the very high trip times, which underlines the influence of large fluctuations. Also for sim-80, the distribution resulting from the simulation does not perfectly match the corresponding modified census distribution. Nevertheless, the effect of large fluctuations due to congestion is smaller than for sim-100. These erratic occurrences of large trip times are also the reason why the reduction of the desired trip times by 20% leads to a decrease in actual trip times by 50%: In sim-100, the system is simply not capable to find a solution that is able to match the demand, and thus has too few contributions at trip times around 500 secs while it has too many contributions at trip times above 3000 secs.

As mentioned above, we do not claim that the 80% census trip time distribution leads to a realistic representation of the real traffic flows in the study area. The idea is just to check the assumption that a reduced distribution leads to more realistic traffic flow patterns. The comparison with the field data is topic of the following section.

36.6 Comparison to field data and to emme/2 study results

First, the field count data is compared with the results of our simulation runs directly for every link. For comparison, the results of the "EMME/2 study" are also shown. Fig. 36.4 shows the typical scatterplots, with field data on the x-axis and simulation results for the same links on the y-axis. Note that both axes are logarithmic.

The first observation is that the plots look remarkably similar in structure. All three studies give relatively unbiased results for high flows, and underestimate low volumes. In addition, there are a few data points where simulation and reality are rather far apart.

At closer inspection, one notes that EMME/2 is somewhat overestimating high volumes, whereas our simulations are underestimating them. This is confirmed by bias calculations (see below). Such an effect is consistent with what one would expect: The Portland Metro assignment model for the presented results does not have a flow cutoff at capacity, so that it is possible to actually put more flow on a link than that link has capacity. This happens in particular at bottlenecks on short links in an otherwise relatively uncongested area.[3] The queue model traffic simulation tends to behave in the opposite way. If demand is higher than capacity, the queue spills back. Once this queue reaches another intersection, that intersection will normally be blocked for all directions, not just for the direction into the congested link. This is a consequence of the fact that the queue model neglects multi-lane effects at intersections. This means, for instance, that a car waiting for a chance to make a left turn blocks all following cars on this link. This tends to cause unrealistically large spill backs.

When one compares sim-80 to sim-100, the flows for sim-80 are closer to the field data for high volumes, and farther away for medium volumes. It is striking that demand reduction by as much as 20% changes the resulting flows so little. This adds to the conjecture that measured flows in a network depend as much on the network structure as on the demand structure.

For more detailed information, one can look at links in different classes regarding field data and direction (Table 36.1). For each class c we calculated the mean absolute and relative bias, i.e.

$$b_{abs,c} = (1/N_c) \sum_i (x_i - \xi_i) = (1/N_c) \left(\sum_i x_i - \sum_i \xi_i \right) \quad \text{and} \quad b_{rel,c} = b_{abs,c} \,/\, \langle \xi \rangle_c \,, \tag{36.9}$$

the mean deviation from the field data, i.e.

$$d_{abs,c} = (1/N_c) \sum_i |x_i - \xi_i| \quad \text{and} \quad d_{rel,c} = d_{abs,c} \,/\, \langle \xi \rangle_c \,, \tag{36.10}$$

[3]This really depends on the cost function which is used. Most cost functions set link speed v to a very low number (but not to zero) at high volumes. Since link costs are proportional to L/v, where L link length, one has that congested links do not contribute much to the cost of a route as long as these links are short and rare. In consequence, much too high volumes can be assigned to such links.

and the root mean square deviation from the field data, i.e.

$$var_c = \left((1/N_c) \sum_i (x_i - \xi_i)^2 \right)^{1/2} \quad \text{and} \quad \sigma_c = var_c \,/\, \langle \xi \rangle_c \,. \tag{36.11}$$

Links were classified by visual inspection into links leading towards the Portland downtown area, and all other links. The tables show that our simulations are underestimating the flows on the "other" links more than they are underestimating the flows on the links towards downtown. Visual inspection of the simulations reveals that this is probably a result of too *much* demand (and thus congestion) for traffic away from the downtown area. This is what one would expect from our simplifications: We are assuming a spatially homogeneous trip time distribution; yet, one would expect that people who live downtown moved there because they have a higher dislike of long trip times than the average population.

Regarding the size classes, sim-100 systematically underestimates volumes except for class 1 (< 250). Sim-80 underestimates less for class 6 (> 1500), underestimates more for all intermediate classes, and is nearly unbiased for class 1. The interpretation of this is that in sim-100, traffic on the major roads is so congested that the routes are pushed onto the smaller streets. The EMME/2 studies, in contrast, systematically over-estimate volumes. Similar to our results, the ratio of traffic on small vs traffic on large roads is too high. Quite possibly, the fastest path search that is used in both approaches makes simulated travelers accept complicated detours on minor streets more easily than in the real world.

Last, one should also remember that the estimated error of the field counts is assumed to be no better than $\pm 15 - 20\%$. We will come back to this point in the discussion.

In summary, one can say the following: Our simulations are far enough progressed to allow tentative comparisons to real world volume counts. The simulations done for this investigation lead to traffic flows with volumes that are somewhat low when compared to reality. Due to the complexity of the approach, there can be many reasons for this, and the systematic analysis of these effects should be the subject of future research.

36.7 Discusssion

The purpose of this study was to couple a simple demand generation method with route assignment and transportation micro-simulation via a computational feedback procedure. We wanted to explore in how far such an approach is feasible, and then out of scientific curiosity and as a benchmark we compared the results with real world data and with existing EMME/2 study results for the same problem. What can one learn from this?

First, it is now indeed both methodologically and computationally possible to systematically couple demand generation, route selection, and transportation micro-simulation. Again, this does not automatically mean that this is always the best method; however, it can and thus should be explored as one of many alternatives. Also note again that practitioners have always done some version of this feedback: If an assignment did not generate plausible flows, it was common practice to adjust the trip matrix (K. Cervenka, personal communication). The main differences thus are that we do it systematically and computerized, and that we use a micro-simulation instead of a static assignment. — The second result is that for the morning peak, extremely simple assumptions yield results which are comparable to results of an EMME/2 study.

An important task would be to separate the influences of the different modules. In addition to the input data, there are four computational modules involved in this study: demand generation, routing, traffic flow simulation, and feedback mechanism. All of these

class	n	mean bias	mean err	RMS err
total	495	-195 (-20%)	342 (36%)	611 (63%)
to-downtown	142	-166 (-15%)	313 (29%)	473 (44%)
other	353	-207 (-23%)	354 (39%)	658 (72%)
< 250	104	46 (32%)	129 (90%)	186 (130%)
$250 - 500$	126	-51 (-14%)	184 (50%)	226 (61%)
$500 - 750$	87	-96 (-15%)	226 (37%)	278 (45%)
$750 - 1000$	44	-184 (-21%)	285 (33%)	367 (43%)
$1000 - 1500$	62	-274 (-23%)	382 (32%)	512 (43%)
> 1500	71	-855 (-25%)	1068 (31%)	1428 (41%)

class	n	mean bias	mean err	RMS err
total	495	-209 (-22%)	344 (36%)	556 (58%)
to-downtown	142	-191 (-18%)	366 (34%)	575 (53%)
other	353	-216 (-24%)	335 (37%)	548 (60%)
< 250	104	2 (1%)	117 (82%)	167 (116%)
$250 - 500$	126	-83 (-23%)	200 (54%)	241 (65%)
$500 - 750$	87	-171 (-28%)	263 (43%)	307 (50%)
$750 - 1000$	44	-212 (-25%)	291 (34%)	370 (43%)
$1000 - 1500$	62	-308 (-26%)	388 (32%)	510 (42%)
> 1500	71	-684 (-20%)	1011 (29%)	1249 (36%)

class	n	mean bias	mean err	RMS err
total	495	83 (9%)	275 (29%)	413 (43%)
to-downtown	142	215 (20%)	318 (29%)	476 (44%)
other	353	30 (3%)	258 (28%)	385 (42%)
< 250	104	84 (59%)	146 (102%)	259 (181%)
$250 - 500$	126	71 (19%)	199 (54%)	263 (71%)
$500 - 750$	87	57 (9%)	212 (34%)	297 (48%)
$750 - 1000$	44	106 (12%)	314 (36%)	376 (44%)
$1000 - 1500$	62	147 (12%)	364 (30%)	473 (39%)
> 1500	71	73 (2%)	574 (16%)	757 (22%)

Table 36.1: TOP: sim-100. MIDDLE: sim-80. BOTTOM: EMME/2 study.

can contribute to variations in the volumes. A systematic study would vary or switch these modules one by one and establish the effect on the volumes. This was beyond the scope of this investigation; the following paragraphs will discuss some of the issues.

NETWORK DATA: We have used the same network input data as the EMME/2 studies. Errors here should, to a certain extent, show up similarly with both approaches. It seems that at the level of current accuracy, there are no major errors in these files. That belief is reinforced by the fact that Portland Metro has been using these files for many years.

DEMAND GENERATION INPUT DATA: The data used here was: household locations, workplace locations, and distributions of start times and trip times. The accuracy of these is unkown. With regard to trip times, it was already discussed earlier that the trip times from the census most probably over-estimate times on our network, for two reasons: (1) Travelers intuitively report the time from door to door, not the time actually on the road. (2) Since many local streets are missing in our network, the time spent in our network should be smaller than the complete time on the road. Indeed, reducing all trip times to 80% ("sim-80") in our study did not lead to significant changes in volumes and even led to *higher* (and more realistic) volumes on the major streets, adding to the assumption that reported trip times are probably too high. Also, just looking at home-to-work trips is a simplification. Any traffic besides home-to-work trips is neglected, such as deliveries, people returning from night shifts, shopping, leisure, etc. All these will be indispensable in order to understand 24-hour traffic patterns.

VOLUME COUNT DATA: There is a slight inconsistency between the input data and the volume count data: Input relies on the census, which is from 1990, while the volume counts are from 1992 and 1994. In fact, the average change (mean bias; see above for definition) of traffic flows from 1992 to 1994 is $+4\%$. A bigger challenge is the variability of the data. Fig. 36.5 shows, where available, the counts from 1992 against the counts from 1994. There is strong variability of the counts, and the average absolute difference (mean error, see above for definition) is in fact 31%.[4] This indicates that in future two things need to be done: (1) Field data need to include a measure of variability; and (2) the corresponding variability measure needs to be obtained from simulations.

ROUTING: This study assumes fastest path routing. Most probably, this is only an approximation of what real people do. In fact, both our simulation results and the model results from the Portland Metro study over-state traffic on minor streets, indicating that the simulated travelers are more willing to accept complicated detours than real world travelers. Also, at the moment no other mode of transportation is included. For the Portland case, this should for example lead to an over-estimation of car traffic between downtown locations.

TRAFFIC FLOW SIMULATION (also called network loading): As discussed earlier, our traffic flow simulation (the queue model) underestimates volumes. In contrast, traditional assignment network loading usually over-estimates volumes (depending on the cost function).

A heuristic possibility for progress would be to design a traffic flow simulation with a behavior somewhere in between our queue model and the traditional assignment network loading. A more systematic approach would be to use a more realistic micro-simulation in order to exactly pin-point the deficiencies. In that context, it would be interesting to also look at link speeds in order to decide whether low counts are caused by low traffic or by congestion. This data is easy to extract from the simulations, but it typically does not exist for the field. ITS technology will have a significant impact here.

FEEDBACK: Our feedback method performs slow adaptation based on the previous iteration, similar to fictitious play in game theory. While the result of such an approach is not exactly a Nash Equilibrium, it is assumed to be close.[5] Two aspects need to be considered separately:

- Convergence/uniqueness: If one sees the second-by-second trajectory of the micro-simulation as a point in state space, then the iterations are mappings from that state space into itself (e.g. Bottom, 2000). The way our iterations are set up, they describe a Markov-process in that state space, which means that the iterations eventually reach a steady state with a corresponding steady state density in state space (e.g. Cantarella and Cascetta, 1995). Little is known about the characteristics of this steady state density distribution, for example if it is unique, or how many iterations one would need to be reasonably close to ergodicity. In practice, it seems that route iterations behave in a similar way as traditional steady state assignment, that is, they normally yield, within Gaussian fluctuations, unique results for the traffic on the link level (e.g. Bottom, personal communication; Nagel et al, 1999). We are not aware of results of how this extends to feedback iterations into the trip distribution as considered in this paper.

[4] This number is larger than one would expect from Fig. 36.5. The reason is that many high volume streets were not counted in both years, thus leading to a smaller mean, which leads to a larger relative error.

[5] For certain –much simpler– systems, one can show that many plausible iteration schemes converge towards the same state (Hofbauer and Sigmund, 1998).

- Human behavior: It is well-known that convergence results are used only because they are scientifically well-defined, not because they are realistic. When comparing to field data, one should keep in mind that it is unclear how close real systems are to the converged result.

INHOMOGENEITIES: One aspect already mentioned earlier in the text but that should be stressed again is that our method unrealistically assumes homogeneity of all aspects of the scenario except for traffic. For example, it is assumed that the behavioral function f_{ch} is the same for everybody, and that one can obtain it by averaging both the trip times and the accessibility over the whole population and the whole region. This is clearly a simplifying assumption — for example, one might expect that people living downtown have a stronger dislike of long trip times than the average population.

Another inhomogeneity in the Portland situation stems from the fact that the part of the metro region which is north of the Columbia river, so-called Clark County, is part of the State of Washington, while the rest of Portland is part of the State of Oregon. Many Oregon workers choose to live in Clark County for the lower property taxes and cheaper large-lot housing (an effect of differences in land use policy), despite the congested commute and Oregon income tax. Oregon has one of the highest personal income taxes of the U.S. States, while Washington does not have a State tax on personal income. Oregon personal income tax is also paid by non-Oregon residents as long as they work in Oregon. Thus, there is a substantial tax incentive for those who live in Clark County to also work there. This, however, is often not possible due to a low jobs-housing ratio in Clark County. All this results in a relatively high split between peak and non-peak direction volumes on the Columbia River bridges. Sales tax is the opposite: There is no sales tax in Oregon while sales taxes in Clark county average 8%. In consequence, retail activity in Clark County is somewhat suppressed by residents' proximity to tax-free shopping in Oregon. For example, there is a major big-box retail area on the Oregon side of the I-5 bridge that owes its existence to the sales tax disparity. (Bill Stein, Portland Metro, personal communication)

This should result in less traffic northbound into Clark county in the morning peak in reality than in our model. This is easy to check since there are only two bridges across the Columbia river. Indeed, with sim-80 we obtain 7473 veh/hour northbound as opposed to 4650 in the field, while southbound the numbers are comparable: 10052 and 9740, respectively. Sim-100 numbers are lower than sim-80 numbers, due to congestion in the model, but have the same tendency.

36.8 Summary

We have implemented a computational feedback between demand generation and traffic simulation in a real world setting in Portland/Oregon. This was done via a double relaxation loop: an inner loop for relaxation of the route assignment with fixed demand, and an outer loop for relaxation of the demand. Typically, about 70 runs of the traffic micro-simulation are necessary for one relaxed result. We have used data from Portland/Oregon.

For simplicity, we have concentrated on assigning workplaces to workers (whose home locations were given). The challenge was to perform this workplace assignment self-consistently such that the resulting trip times correspond to the trip time distribution given via census data.

Our results demonstrate that with current computational technology and simple models, it is possible to do such studies while retaining microscopic resolution throughout the whole computation. Microscopic resolution here means that each of the about 500 000 travelers and each vehicle are represented individually in each step of the method.

Our simulations were run on single CPU workstations; one relaxation series typically took about four days of computer time.

Because of the many simplifications, we did not expect our results to be a good model of reality. Nevertheless, in order to provide a benchmark we compared our results to real world morning peak volume counts from the Portland/Oregon area, and we included into the comparison results of an older study by Portland Metro using different methods. These results are summarized in Fig. 36.4. It is encouraging that one gets so close with so relatively little investment in terms of input data. In fact, input data consists of nothing more but the EMME/2 street network information, some population characteristics from the census (home locations of workers; overall trip time distribution for home-to-work trips; overall trip starting time distribution), and the locations of workplaces. The methodology uses a relaxation algorithm of workplace assignment, a fastest-path routing, and a queuing micro-simulation. Our study demonstrates that such a microscopic approach is both computationally and methodologically feasible even on modest computing hardware.

36.9 Acknowledgments

We are extremely grateful to B. Stein, D. Walker, K. Lawton, and others at Portland Metro for providing the data for the Portland/Oregon area, without which this study would not have been possible at all. Much of the work was done while the authors were at Los Alamos National Laboratory (LANL) and at Santa Fe Institute (SFI). We thank the Transims project at LANL for providing the technical infrastructure necessary for running these studies.

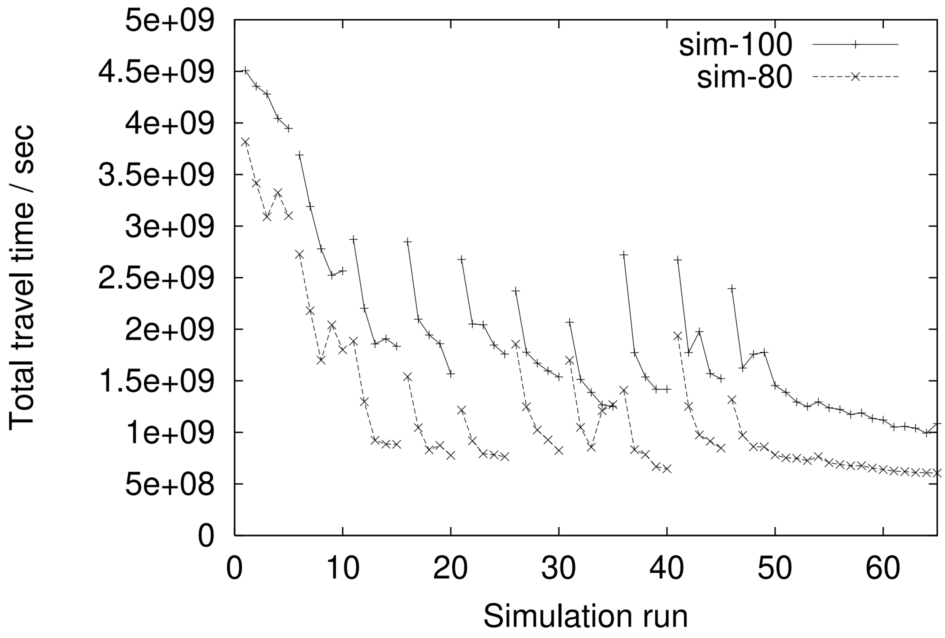


Figure 36.2: Total trip time in the simulation during the iterative assignment with the original census trip time distribution (sim-100) and the census distribution with trip times reduced to 80% (sim-80).

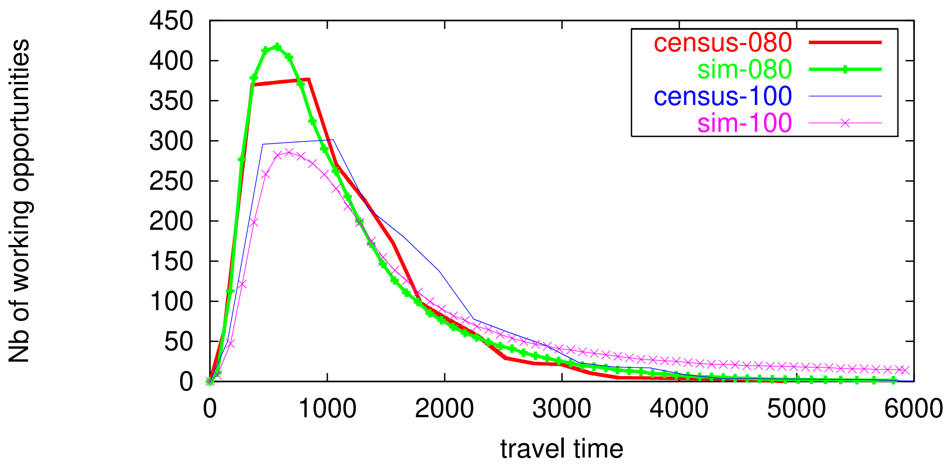


Figure 36.3: Trip time distributions in the queuing simulation at the 70th iteration in comparison to the 100% and the 80% census trip time distribution. Only completed trips contribute to the distribution.

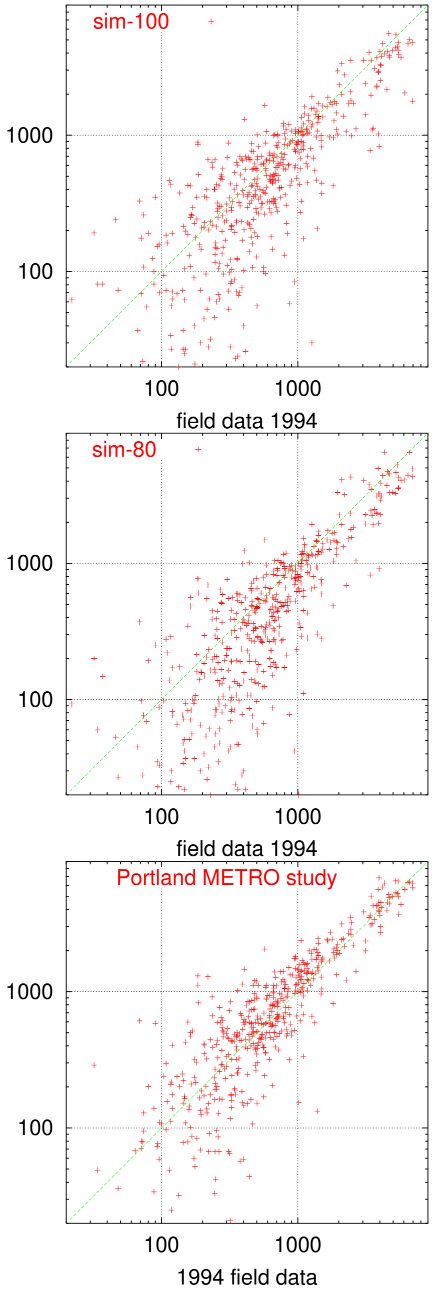


Figure 36.4: Scatterplot of simulated data (y-axis) vs. field data (x-axis). TOP: sim-100. CENTER: sim-80. BOTTOM: EMME/2-study. It is remarkable that reducing the desired trip times by 20% (top to middle) does not seem to change very much at all.

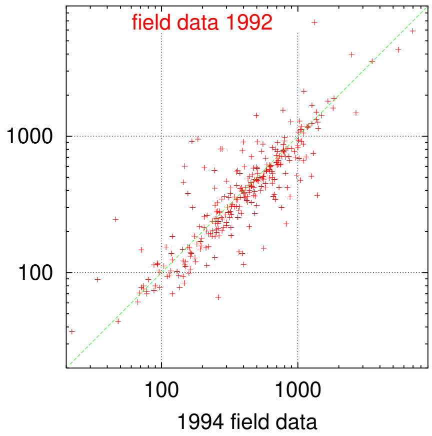


Figure 36.5: Variability of field data. For some measurement locations, count data were available both for 1994 and 1992. For those locations, the 1992 value is plotted against the 1994 value. A better understanding of field data variability will be necessary for further progress.

Chapter 37

A Switzerland case

Acknowledgments

Los Alamos National Laboratory makes the Transims software available to academic institutions for a small charge.

The Swiss Federal Administration provides the input data for the Switzerland studies.

Res Voellmy, Nurhan Cetin, Bryan Raney, Nicolas Lefebvre, Roger Ruegg, Adrian Burri.

Kay Axhausen.

Bibliography

PhD thesis.

T.A. Arentze, F. Hofmann, C.H. Joh, and H.J.P. Timmermans. Experiences with developing ALBATROSS: A learning-based transportation oriented simulation system. In *Verkehr und Mobilität*, number 66 in Stadt Region Land, pages 61–70. Institut für Stadtbauwesen, Technical University, Aachen, Germany, 1998.

K.W. Axhausen. A simultaneous simulation of activity chains. In P.M. Jones, editor, *New Approaches in Dynamic and Activity-based Approaches to Travel Analysis*, pages 206–225. Avebury, Aldershot, 1990.

A. Babin, M. Florian, L. James-Lefebvre, and H. Spiess. EMME/2: Interactive graphic method for road and transit planning. *Transportation Research Record*, 866:1–9, 1982.

M. Bando, K. Hasebe, A. Nakayama, A. Shibata, and Y. Sugiyama. Structure stability of congestion in traffic dynamics. *Japan Journal of Industrial and Applied Mathematics*, 11(2):203–223, 1994.

M. Bando, K. Hasebe, A. Nakayama, A. Shibata, and Y. Sugiyama. Dynamical model of traffic congestion and numerical simulation. *Phys. Rev. E*, 51(2):1035–1042, 1995.

R. Barlovic, L. Santen, A. Schadschneider, and M. Schreckenberg. Metastable states in CA models for traffic flow. *European Physical Journal B*, 5(3):793–800, 1998.

C. L. Barrett, Personal communication.

C. L. Barrett, S. Eubank, K. Nagel, J. Riordan, and M. Wolinsky. Issues in the representation of traffic using multi-resolution cellular automata. Los Alamos Unclassified Report (LA-UR) 95-2658, Los Alamos National Laboratory, Los Alamos, NM, U.S.A., see www.lanl.gov, 1995.

C. L. Barrett, R. Jacob, and M. V. Marathe. Formal-language-constrained path problems. *SIAM J COMPUT*, 30(3):809–837, 2000.

C. L. Barrett, M. Wolinsky, and M. W. Olesen. Emergent local control properties in particle hopping traffic simulations. In D.E. Wolf, M. Schreckenberg, and A. Bachem, editors, *Traffic and granular flow*, pages 169–173. World Scientific, Singapore, 1996.

R. J. Beckman, K. A. Baggerly, and M. D. McKay. Creating synthetic base-line populations. *Transportion Research Part A – Policy and Practice*, 30(6):415–429, 1996.

R.J. Beckman et al. TRANSIMS–Release 1.0 – The Dallas-Fort Worth case study. Los Alamos Unclassified Report (LA-UR) 97-4502, Los Alamos National Laboratory, Los Alamos, NM, see transims.tsasa.lanl.gov, 1997.

M. Ben-Akiva. Route choice models. Presented at the Workshop on “Human Behaviour and Traffic Networks”, Bonn, December 2001.

M. Ben-Akiva and S. R. Lerman. *Discrete choice analysis*. The MIT Press, Cambridge, MA, 1985.

J.A. Bottom. *Consistent anticipatory route guidance*. PhD thesis, Massachusetts Institute of Technology, Cambridge, MA, 2000.

J. L. Bowman. *The day activity schedule approach to travel demand analysis*. PhD thesis, Massachusetts Institute of Technology, Cambridge, MA, 1998.

M. Bradley. A system of activity-based models for Portland, Oregon, Draft final report, 1997.

W. Brilon and N. Wu. Evaluation of cellular automata for traffic flow simulation on freeway and urban streets. In W. Brilon, F. Huber, M. Schreckenberg, and H. Wallentowitz, editors, *Traffic and Mobility: Simulation – Economics – Environment*, pages 163–180. Springer, Berlin, 1998.

A. Burriad. Intersection dynamics in queue models. Term project report, Swiss Federal Institute of Technology, 2002. See sim.inf.ethz.ch/papers.

B. W. Bush. Personal communication.

G. D. B. Cameron and C. I. D. Duncan. PARAMICS — Parallel microscopic simulation of road traffic. *Journal of Supercomputing*, 10(1):25, 1996.

C. Cantarella and E. Cascetta. Dynamic process and equilibrium in transportation network: Towards a unifying theory. *Transportation Science A*, 25(4):305–329, 1995.

E. Cascetta, D. Inaudi, and G. Marquis. Dynamic estimators of origin-destination matrices using traffic counts. *Transportation Science*, 27(4):363–373, 1993.

E. Cascetta and A. Papola. An implicit availability/perception random utility model for path choice. In *Proceedings of TRISTAN III*, volume 2, San Juan, Puerto Rico, 1998.

M.J. Cassidy and J. Han. Validation and evaluation of freeway simulation models. final report. Technical Report FHWA/CA/Purdue-RR-95-1, Purdue University, School of Civil Engineering, West Lafayette IN 47907, USA, 1995.

I. Chabini. Discrete dynamic shortest path problems in transportation applications: Complexity and algorithms with optimal run time. *Transportation Research Record*, 1645: 170–175, 1998a.

I. Chabini. Discrete dynamic shortest path problems in transportation applications: Complexity and algorithms with optimal run time. In *Transportation Research Record* (Chabini, 1998a), pages 170–175.

G.L. Chang, H.S. Mahmassani, and R. Herman. A macroparticle traffic simulation model to investigate peak-period commuter decision dynamics. *Transportation Research Record*, 1005:107–120, 1985.

D. Chowdhury, L. Santen, and A. Schadschneider. Statistical physics of vehicular traffic and some related systems. *Physics Reports*, 329(4–6):199–329, May 2000.

D. Chowdhury, L. Santen, A. Schadschneider, S. Sinha, and A. Pasupathy. Spatio-temporal organization of vehicles in a cellular automata model of traffic with 'slow-to-start' rule. *J. Physics A: Math. General*, 32:3229, 1999.

S. Clarke, A. Krikorian, and J. Rausen. Computing the n best loopless paths in a network. *J. Soc. Indust. Appl. Math.*, 11(4):1096–1102, December 1963.

M. Cremer and M. Papageorgiou. Parameter identification for a traffic flow model. *Automatica*, 17(6):837–843, 1981.

M. Cremer and H. Schütt. A comprehensive concept for simultaneous state observation, parameter estimation, and incident detection. In *Proceedings of the 11th Int. Symposium on Transportation and Traffic Theory*, Yokohama, Japan, 1990.

Carlos F. Daganzo, M. J. Cassidy, and R. L. Bertini. Possible explanations of phase transitions in highway traffic. *Transportation Research A*, 33:365–379, 1999.

R.W. Denney, J.C. Williams, S.C.S. Bhat, and S.A. Ardekani. Calibrating NETSIM for a CBD using the two fluid model. In *Large Urban Systems. Proceedings of the Advanced Traffic Management Conference*. Federal Highway Administration, 400 7th Street SW, Washington DC, USA, 1993.

S. T. Doherty and K. W. Axhausen. The developement of a unified modelling framework for the household activity-travel scheduling process. In *Verkehr und Mobilität*, number 66 in Stadt Region Land. Institut für Stadtbauwesen, Technical University, Aachen, Germany, 1998.

Th. A. Domencich and D. McFadden. Urban travel demand. In D.W. Jorgenson and J. Waelbroeck, editors, *Urban travel demand*, number 93 in Contributions to Economic Analysis. North-Holland and American Elsevier, 1975.

J.J. Dongarra, I.S. Duff, D.C. Sorensen, and H.A. van der Vorst. *Numerical linear algebra for high-performance computers*. Software, Environments, and Tools. SIAM Society for Industrial and Applied Mathematics, Philadelphia, 1998.

DYNAMIT. See its.mit.edu and dynamictrafficassignment.org, accessed 2003.

DYNASMART. See www.dynasmart.com and dynamictrafficassignment.org, accessed 2003.

J. Esser. *Simulation von Stadtverkehr auf der Basis zellularer Automaten*. PhD thesis, University of Duisburg, Germany, 1998a.

J. Esser. *Simulation von Stadtverkehr auf der Basis zellularer Automaten*. PhD thesis, University of Duisburg, Germany, 1998b. See also www.traffic.uni-duisburg.de.

J. Esser and K. Nagel. Census-based travel demand generation for transportation simulations. In W. Brilon, F. Huber, M. Schreckenberg, and H. Wallentowitz, editors, *Traffic and Mobility: Simulation – Economics – Environment*, pages 135–148, Berlin, 1998. Springer.

U. Frisch, B. Hasslacher, and Y. Pomeau. Lattice-gas automata for navier-stokes equation. *Phys. Rev. Letters*, 56:1505, 1986.

C. Gawron. An iterative algorithm to determine the dynamic user equilibrium in a traffic simulation model. *International Journal of Modern Physics C*, 9(3):393–407, 1998a.

C. Gawron. An iterative algorithm to determine the dynamic user equilibrium in a traffic simulation model. *International Journal of Modern Physics C*, 9(3):393–407, 1998b.

D. L. Gerlough and M. J. Huber. *Traffic Flow Theory*. Special Report No. 165. Transportation Research Board, National Research Council, Washington, D.C., 1975.

P. G. Gipps. A behavioural car-following model for computer simulation. *Transportation Research B*, 15:105–111, 1981.

P. G. Gipps. A model for the structure of lane-changing decisions. *Transportation Research B*, 20B(5):403–414, 1986.

C. Gloor. Modelling of autonomous agents in a realistic road network (in German). Diplomarbeit, Swiss Federal Institute of Technology ETH, Zürich, Switzerland, 2001.

R. Haberman. *Mathematical models in mechanical vibrations, population dynamics, and traffic flow*. Prentice-Hall, Englewood Cliffs, NJ, 1977.

D. Helbing. *Verkehrsdynamik*. Springer, Heidelberg, Germany, 1997.

R. Herman and I. Prigogine. A two-fluid approach to town traffic. *Science*, 204:148–151, 1979.

J. Hofbauer and K. Sigmund. *Evolutionary games and replicator dynamics*. Cambridge University Press, 1998.

J.D. Holland. *Adaptation in Natural and Artificial Systems*. Bradford Books, 1992. Reprint edition.

R. R. Jacob, M. V. Marathe, and K. Nagel. A computational study of routing algorithms for realistic transportation networks. *ACM Journal of Experimental Algorithms*, 4 (1999es, Article No. 6), 1999.

A. Jakobs and R.W. Gerling. Scaling aspects for the performance of parallel algorithms. *Parallel Computing*, 19(9):1063–1073, 1993.

D. Jost and K. Nagel. Probabilistic traffic flow breakdown in stochastic car following models. Paper 03-4266, Transportation Research Board Annual Meeting, Washington, D.C., 2003. Shorter version to be published in Transportation Research Records.

T. Kelly. Driver strategy and traffic system performance. *Physica A*, 235:407, 1997.

B. S. Kerner. Traffic flow: Experiment and theory. In D.E. Wolf and M. Schreckenberg, editors, *Traffic and granular flow'97*, pages 239–267. Springer, Berlin, 1998.

B. S. Kerner and P. Konhäuser. Structure and parameters of clusters in traffic flow. *Phys. Rev. E*, 50(1):54–83, 1994.

B. S. Kerner and H. Rehborn. Experimental features and characteristics of traffic jams. *Phys. Rev. E*, 53(2):R1297–R1300, 1996a.

B. S. Kerner and H. Rehborn. Experimental properties of complexity in traffic flow. *Phys. Rev. E*, 53(5):R4275–R4278, 1996b.

J.H. Kim. Special issue about the first micro-robot world cup soccer tournament, MIROSOT. *Robotics and Autonomous Systems*, 21(2):137–205, 1997.

S. Krauß. *Microscopic modeling of traffic flow: Investigation of collision free vehicle dynamics*. PhD thesis, University of Cologne, Germany, 1997. See www.zaik.uni-koeln.de/~paper.

S. Krauß, K. Nagel, and P. Wagner. The mechanism of flow breakdown in traffic flow models. Technical report, 1998.

S. Krauß, P. Wagner, and C. Gawron. Metastable states in a microscopic model of traffic. *Phys. Rev. E*, 55(5):5597–5602, 1997.

R.D. Kühne and R. Beckschulte. Non-linearity stochastics of unstable traffic flow. In C.F. Daganzo, editor, *Proc. 12th Int. Symposium on Theory of Traffic Flow and Transportation*, page 367. Elsevier, Amsterdam, The Netherlands, 1993.

M. J. Lighthill and J. B. Whitham. On kinematic waves. I: Flow movement in long rivers. II: A Theory of traffic flow on long crowded roads. *Proceedings of the Royal Society A*, 229:281–345, 1955.

D. Lohse. *Verkehrsplanung*, volume 2 of *Grundlagen der Straßenverkehrstechnik und der Verkehrsplanung*. Verlag für Bauwesen, Berlin, 1997.

H.S. Mahmassani, J.C. Williams, and R. Herman. Performance of urban traffic networks. In N.H. Gartner and N.H.M. Wilson, editors, *Transportation and Traffic Theory*, page 1. Elsevier Science Publishing Co., Inc., 1987.

A.D. May. *Traffic flow fundamentals*. Prentice Hall, Englewood Cliffs, NJ, 1990.

P. Metaxatos, D. Boyce, M. Florian, and I. Constantin. Implementing combined model of origin-destination and route choice in EMME/2 system. *Transportation Research Records*, 1493:57–63, 1995.

MITSIM, 1999. Massachusetts Institute of Technology, Cambridge, Massachusetts. See its.mit.edu.

MPI www page. www-unix.mcs.anl.gov/mpi/, accessed 2003. MPI: Message Passing Interface.

K. Nagel. Freeway traffic, cellular automata, and some (self-organizing) criticality. In R.A. de Groot and J. Nadrchal, editors, *Physics Computing '92*, page 419, Prague, 1992. World Scientific.

K. Nagel. Particle hopping models and traffic flow theory. *Phys. Rev. E*, 53(5):4655–4672, 1996.

K. Nagel. From particle hopping models to traffic flow theory. *Transportation Research Records*, 1644:1–9, 1999.

K. Nagel and C.L. Barrett. Using microsimulation feedback for trip adaptation for realistic traffic in Dallas. *International Journal of Modern Physics C*, 8(3):505–526, 1997.

K. Nagel and H. J. Herrmann. Deterministic models for traffic jams. *Physica A*, 199: 254, 1993.

K. Nagel and M. Paczuski. Emergent traffic jams. *Phys. Rev. E*, 51:2909–2918, 1995.

K. Nagel and S. Rasmussen. Traffic at the edge of chaos. In R. A. Brooks and P. Maes, editors, *Artificial Life IV: Proceedings of the Fourth International Workshop on the Synthesis and Simulation of Living Systems*, pages 222–235. MIT Press, Cambridge, MA, 1994a.

K. Nagel and S. Rasmussen. Traffic at the edge of chaos. In R. A. Brooks and P. Maes, editors, *Artificial Life IV: Proceedings of the Fourth International Workshop on the Synthesis and Simulation of Living Systems*, pages 222–235. MIT Press, Cambridge, MA, 1994b.

K. Nagel, M. Rickert, P. M. Simon, and M. Pieck. The dynamics of iterated transportation simulations. See www.arXiv.org, nlin.AO/0002040, 2000. Earlier version in: Proceedings of 3rd TRIannual Symposium on Transportation ANalysis (TRISTAN-III) 1998 in San Juan, Puerto Rico.

K. Nagel and A. Schleicher. Microscopic traffic modeling on parallel high performance computers. *Parallel Computing*, 20:125–146, 1994.

K. Nagel and M. Schreckenberg. A cellular automaton model for freeway traffic. *Journal de Physique I France*, 2:2221–2229, 1992.

K. Nagel, P. Stretz, M. Pieck, S. Leckey, R. Donnelly, and C. L. Barrett. TRANSIMS traffic flow characteristics. Los Alamos Unclassified Report (LA-UR) 97-3530, Los Alamos National Laboratory, Los Alamos, NM, see transims.tsasa.lanl.gov, 1997.

K. Nagel, P. Wagner, and R. Woesler. Still flowing: Approaches to traffic flow and traffic jam modeling. *Operations Research*, 51(5):681–710, 2003.

K. Nagel, D.E. Wolf, P. Wagner, and P. M. Simon. Two-lane traffic rules for cellular automata: A systematic approach. *Phys. Rev. E*, 58(2):1425–1437, 1998.

W. Niedringhaus, J. Opper, L. Rhodes, and B. Hughes. IVHS traffic modeling using parallel computing: Performance results. In *Proceedings of the International Conference on Parallel Processing*, pages 688–693. IEEE, 1994.

J. de D. Ortúzar and L.G. Willumsen. *Modelling transport*. Wiley, Chichester, 1995.

R. Palmer. Broken ergodicity. In D. L. Stein, editor, *Lectures in the Sciences of Complexity*, volume I of *Santa Fe Institute Studies in the Sciences of Complexity*, pages 275–300. Addison-Wesley, Redwood City, CA, 1989.

D. Park and L. R. Rilett. Identifying multiple and reasonable paths in transportation networks: A heuristic approach. *Transportation Research Records*, 1607:31–37, 1997.

Michael Patriksson. *The Traffic Assignment Problem: Models and Methods*. Topics in Transportation. VSP, Zeist, The Netherlands, 1994.

A. Perko. Implementation of algorithms for k shortest loopless paths. *Networks*, 16: 149–160, 1986.

M. Ponzlet and P. Wagner. Validation of a CA–model for traffic simulation of the Northrhine-Westphalia motorway network. In *The 24th European Transport Forum, Proceedings*, volume P404-1, 1996.

PVM www page. www.epm.ornl.gov/pvm/, accessed 2003. PVM: Parallel Virtual Machine.

H. A. Rakha and M. W. Van Aerde. Comparison of simulation modules of TRANSYT and INTEGRATION models. *Transportation Research Record*, 1566:1–7, 1996.

M. Rickert. *Traffic simulation on distributed memory computers*. PhD thesis, University of Cologne, Cologne, Germany, 1998. available via www.zaik.uni-koeln.de/˜paper.

M. Rickert, K. Nagel, M. Schreckenberg, and A. Latour. Two lane traffic simulations using cellular automata. *Physica A*, 231(4):534–550, 1996a.

M. Rickert, K. Nagel, M. Schreckenberg, and A. Latour. Two lane traffic simulations using cellular automata. *Physica A*, 231:534, 1996b.

J.D. Rothwell. *Control of Human Voluntary Movement*. Chapman and Hall, 1994.

G. Sauermann and H.J. Herrmann. A 1d traffic model with threshold parameters. In D.E. Wolf and M. Schreckenberg, editors, *Traffic and granular flow'97*, pages 481–486. Springer, Berlin, 1998.

A. Schadschneider. Analytical approaches to cellular automata for traffic flow: Approximations and exact solutions. In D.E. Wolf and M. Schreckenberg, editors, *Traffic and granular flow'97*, pages 417–432. Springer, Berlin, 1998.

A. Schadschneider and M. Schreckenberg. Cellular automaton models and traffic flow. *J. Physics A: Math. General*, 26:L679, 1993.

T. Schwerdtfeger. *Makroskopisches Simulationsmodell für Schnellstraßennetze mit Berücksichtigung von Einzelfahrzeugen (DYNEMO)*. PhD thesis, University of Karsruhe, Germany, 1987.

Y. Sheffi. *Urban transportation networks: Equilibrium analysis with mathematical programming methods*. Prentice-Hall, Englewood Cliffs, NJ, USA, 1985.

P. M. Simon and K. Nagel. Simple queueing model applied to the city of Portland. *International Journal of Modern Physics C*, 10(5):941–960, 1999.

U. Sparmann. *Spurwechselvorgänge auf zweispurigen BAB–Richtungsfahrbahnen*. Number 263 in Forschung Straßenbau und Straßenverkehrstechnik. Bundesminister für Verkehr, Bonn–Bad Godesberg, Germany, 1978.

D. Sternad. personal communication.

TRANSIMS www page. TRansportation ANalysis and SIMulation System. transims.tsasa.lanl.gov, accessed 2003. Los Alamos National Laboratory, Los Alamos, NM.

Transportation Research Board. *Highway Capacity Manual*. In *Special Report No. 209*, Transportation Research Board (1994b), 3rd edition, 1994a.

Transportation Research Board. *Highway Capacity Manual*. Special Report No. 209. National Research Council, Washington, DC, 3rd edition, 1994b.

H. Unger. An approach using neural networks for the control of the behaviour of autonomous individuals. In A. Tentner, editor, *High Performance Computing 1998*, pages 98–103. The Society for Computer Simulation International, 1998.

H. Unger. *Modellierung des Verhaltens autonomer Verkehrsteilnehmer in einer variablen staedtischen Umgebung*. PhD thesis, TU Berlin, 2002.

M. Van Aerde, personal communication.

M. Van Aerde, B. Hellinga, M. Baker, and H. Rakha. INTEGRATION: An overview of traffic simulation features. 1996. A paper accepted for presentation at the 1996 Transportation Research Board Annual meeting.

J. Van Leeuwen, editor. *Formal models and semantics*, volume B of *Handbook of Theoretical Computer Science*, 1990. Elsevier and MIT Press.

VISSIM www page. www.ptv.de, accessed 2003. Planung Transport und Verkehr (PTV) GmbH.

P. Wagner. Traffic simulations using cellular automata: Comparison with reality. In D E Wolf, M.Schreckenberg, and A.Bachem, editors, *Traffic and Granular Flow*. World Scientific, Singapore, 1996.

P. Wagner and K. Nagel. Microscopic modeling of travel demand: Approaching the home-to-work problem. Paper 99 09 19, Transportation Research Board Annual Meeting, Washington, D.C., 1999.

P. Wagner, K. Nagel, and D.E. Wolf. Realistic multi-lane traffic rules for cellular automata. *Physica A*, 234:687, 1997.

S. Weinmann. *Simulation of spatial learning mechanisms*. PhD thesis, Swiss Federal Institute of Technology ETH, Zürich, Switzerland, in preparation.

R. Wiedemann. Simulation des Straßenverkehrsflusses. Schriftenreihe Heft 8, Institute for Transportation Science, University of Karlsruhe, Germany, 1994.

R. Wiedemann. Beschreibung des Staus. In H. Keller, editor, *Beiträge zur Theorie des Straßenverkehrs*. Forschungsgesellschaft für Straßen- und Verkehrswesen, Köln, Germany, 1995.

D.E. Wolf. Cellular automata for traffic simulations. *Physica A*, 263:438–451, 1999.

S. Wolfram. *Theory and Applications of Cellular Automata*. World Scientific, Singapore, 1986.

www-users.cs.umn.edu/˜karypis/metis/. METIS library, accessed 2003.

Yin Y. Yen. Finding the k shortest loopless paths in a network. *Management Science*, 17 (11):712–716, July 1971.

S. Yukawa and M. Kikuchi. Coupled-map modeling of one-dimensional traffic flow. *Journal of the Physical Society of Japan*, 64(1):35–38, 1995.