Manual

Microsimulation Suite Pamina III

Version 0.9.1

Dr. Marcus Rickert

May 2005

Contents

Chapter 1

Introduction

Acknowledgements

The following people have directly or indirectly contributed to PAMINA and/or this manual:

- Prof. Dr. Achim Bachem
- Chris Barrett
- Ken Cervenka (North Central Texas Council of Governments)
- Dr. Christian Gawron
- Terence Kelly
- Andreas Latour
- Prof. Dr. Kai Nagel
- Dr. Thomas Pfenning
- Dr. Frank Meisgen
- Jack Morrison
- Martin Pieck
- Steen Rasmussen
- Prof. Dr. Rainer Schrader
- Prof. Dr. Michael Schreckenberg
- Prof. Dr. Ewald Speckenmeyer

- Prof. Dr. Dietrich Stauffer
- Paula Stretz
- Dr. Peter Wagner

Thank you very much!

Foreword

Note to the impatient reader: This chapter mainly covers background information on traffic simulation. If you are more interested in the actual usage of the simulator PAMINA, please, proceed to chapter 2 on page 6.

Most of the material presented in this manual was drawn from the Ph.D. thesis of the author. For a more detailed discussion of all topics presented here, please refer to [40] which is available at

`http://www.the-rickerts.de/mr/dissertation.en.html` .

The author gladly welcomes any comments on the this manual or the thesis. Please, be aware that the thesis was written in 1998, so terms such as 'now', 'recently', or 'in the future' have to seen as relative to this.

Traffic Simulation

Recently, there has been increased interest in microscopic traffic simulations worldwide. In contrast to earlier implementations which could only handle small street networks in reasonable time, current state-of-the-art implementations exploit the architecture of modern computer systems to increase their performance considerably. Also, there has been a shift away from macroscopic underlying traffic models to simple microscopic ones such as the cellular automaton approach.

In this chapter we start out by giving a short overview of existing micro-simulations that are able to execute route-sets within regional street networks.

1.1 Simulation models

Currently, there are several commercial and non-commercial traffic simulation packages available. Some [34, 35] are based on macroscopic traffic models, which neglect individual characteristics of vehicles including route-plans. Although their computational performance is usually higher than microscopic models, they lack the ability to run activity-based simulations based upon route-plans.

In the following, we will outline those micro-simulations capable of executing individual route-plans. For a more comprehensive survey of micro-simulations see [48].

NETSIM [37] was originally developed for the Federal Highway Administration. Later, it was integrated with TRAF simulations system resulting in the new common name TRAF-NETSIM. It puts special emphasis on handling stochasticity of driver decisions correctly, since random driver behavior at the microscopic level is known to have a considerable impact on aggregated measurements. In one test-bed [20] the whole core street network of Austin (TX) was simulated on a CRAY vector computer.

INTEGRATION [1, 15] is a microscopic traffic-simulation developed at the Queen's University in Kingston, Canada. Its dynamics are based upon a car-following model with a macroscopic calibration for the desired free speed, the speed at capacity, and the jam density of each link. The network capabilities of the simulation include among other things lane-changing, incidents, freeway intersections, turning movement restrictions, traffic signals, loop detectors, and vehicle probes.

DYNASMART (=Dynamic Network Assignment Simulation Model for Advanced Road Telematics, see [9, 18]) is a micro-simulation driven by individual route-plans. It provides the capability to explicitly model trip maker en-route decisions in response to online information.

DYNEMO [45] represents a special case within this list of traffic-simulations. In principle, it is a *macroscopic* model, since it uses segmented links with link performance functions. The necessary input for these functions such as density and mean velocity, however, is retrieved by aggregating over *individual* vehicles. Therefore, DYNEMO is capable of processing route-plans as every other model presented here.

TRANSIMS [30, 47] is a traffic research project funded by the American Federal Highway Administration [32]. It comprises modules to (a) generate a synthetic population from census data, (b) generate activities from the synthetic population, and (c) generate a route-set from the activities. The simulation itself, which is also based upon the Nagel-Schreckenberg model, was parallelized using workstation clusters with a distributed memory programming model.

The traffic research effort FVU-NRW [31] funded by the German federal state Nordrhein-Westfalen uses a modular traffic simulation called PLANSIM-T [17]. Its structure corresponds to that of TRANSIMS. Since privacy laws in Germany largely restrict access to census data for research purposes, route-sets are more likely to be obtained from origin-destinations matrices. The matrix is computed from homogeneous groups of population distributed over the simulation area. Actual traffic counts are used to calibrate the OD-flows, which currently still poses a problem [49]. The micro-simulation features three built-in underlying traffic models: (a) the original Nagel-Schreckenberg model [36], (b) a refined continuous model described in [19], and (c) a low resolution queuing model, which replaces incoming lanes by queues. PLANSIM-T uses the thread programming paradigm on shared memory multi-processor computers.

The city traffic simulation CASim [8, 14] developed at the university of Duisburg, Germany (also within the framework of the FVU-NRW) uses the original Nagel-Schreckenberg CA. The simulation can be driven by both turn counts at intersections and individual route-plans. Furthermore, it can be influenced by online traffic count data which is used to perform dynamic re-routing.

feature	PAMINA I	PAMINA II	PAMINA III
individual speed limit	no	no	yes
transfer rates	yes	no	no
route-plans	no	random	yes
traffic signals	no	no	yes
message passing library	PVM	PVM	PVM/MPI
parallelization	direct	Toolbox 1.0	Toolbox 2.0
off-line load balancing	yes	yes	yes
off-line feedback	no	no	yes
online load balancing	no	yes	no

Table 1.1: *Overview over PAMINA versions*

The PARAMICS [7, 33] simulation developed at the Edinburgh Parallel Computing Center uses (in its original version) a Connection Machine CM-200. New releases have been ported to message-passing systems like the CRAY T3D/E.

1.2 Overview of PAMINA versions

Using the CA model developed by Nagel and Schreckenberg as a starting point, we extended the model to include traffic in street networks. The goal was to maintain the simplicity of the CA model as far as possible by using a few building blocks based upon the original model. We have implemented three versions (see Table 1.1) of the micro-simulation PAMINA which we would like to describe in this chapter.

Note that only the most current version of the traffic simulator (PAMINA III) is available.

The first version PAMINA I [38] already used the multi-lane extension of the Nagel-Schreckenberg CA to simulate traffic on links. Vehicles are generated using sources with time-dependent insertion rates. At each of the traffic nodes (e.g. junction or ramp) vehicles are transferred to their new respective destination links according to time-dependent transfer rates. In this respect it is similar to CASim. There is, however, no calibration of the transfer rates as in the traffic simulation of the city traffic of Duisburg. Vehicles are removed from the network at sinks with time-dependent absorption rates. The implementation used a distributed memory approach with PVM as the message passing library.

The second version PAMINA II focused on the computational aspects of load-balancing. In contrast to PAMINA I, which used a static load-balancing scheme, PAMINA II used online measurements of execution time to balance the computational load on all CPUs of the parallel computer system. It was also used as a feasibility proof for the simulation of the whole Autobahn network of Germany in real-time [43].

The third and current version PAMINA III shifted the focus to the actual application of the traffic simulation. The network model was extended to include simple signalized intersections. Also,

vehicles now follow individual routes on their trips through the network instead of obeying transfer-rates or random routes. PAMINA III (or simply PAMINA in the following) will be described in detail in the following chapter.

The main objective of PAMINA is to execute a route-set (a list of route-plans) in a street network. Each route-plan is defined by a source, a destination, a list of intermediate net points, and a departure time. After a vehicle has been instantiated[1] at the given departure time, it will be inserted into the simulation network at the origin. It will then execute the route-plan until it reaches its destination. Finally, it will be removed from the system after statistics about its actual travel time have been collected.

A simulation run is initiated by supplying a map defining the geometry of the street network and the route-set. Vehicles will be instantiated according to their departure times until all routes have been processed. The simulation will continue until a given percentage of all instantiated vehicles have reached their destination. For the simulation runs presented in the example section (2.4) of the next the simulation time is set to a fixed interval (here 7 hours).

[1]In object-oriented programming languages the term *instantiate* is used for the dynamic creation of a memory object (e.g. vehicle). These objects usually have a limited life-time before they are *deleted* or *destroyed*.

Chapter 2

Simulator Pamina III

2.1 Network elements

One of the first applications of the network simulation PAMINA was to simulate the whole Autobahn network of Germany in *real-time*, which is to say that one simulation second takes as long as one wall-clock second. The network was given as a graph with nodes and links which had to be represented in the simulation. In the following sections we will describe this network representation.

The network representation used in PAMINA is a graph in which each intersection is represented by a node[1] and each street segment between intersections corresponds to two edges[2]. Moreover, there are nodes defined by the *natural* boundaries of a road network with node degree one, called *terminators*, and additional nodes with degree two where vehicles can enter or exit the network, called *ramps*[3]. This network is usually supplied in two sets of objects: (a) the *set of nodes*, each of which has a unique number (id) and the geometric location of the object, given in rectangular coordinates relative to an arbitrary, but fixed point, and (b) the *set of edges*, each of which has two references (by id) to nodes and optional information such as name, number of lanes, or speed limit.

2.1.1 Street segments

The directed connection (edge or link) between two nodes is represented as a grid equivalent to the model by Nagel/Schreckenberg and its two-lane extension. The characteristics *length*[4], *speed limit*, and *number of lanes* are used to adapt the CA model. The size of the grid is computed by using the grid-site length of 7.5 [meter] as a unit.

[1] Also known as *vertex*.

[2] A segment can correspond to one edge or two edges depending on whether the two directions are equivalent, or not. PAMINA uses the latter, even if all characteristics of both directions are identical, since this symmetry is broken during simulation, anyway.

[3] The segments feeding the ramps are not part the network. Therefore they do not increase the degree of a ramp. If the map were extended to include lower hierarchies, ramps would also have degrees larger than two.

[4] The length of a street segment is either explicitly given or derived from the Euclidean distance of the two nodes.

It is important to note that the characteristics mentioned so far are *constant* for the whole segment. Typical details like additional turning lanes in front of intersections may be modeled by inserting additional nodes to split a given segment and assigning different parameters to the various parts.

For the convenience of the reader we would like to outline the single lane CA model[5] introduced by Nagel and Schreckenberg. The system consists of a one dimensional grid of L sites with periodic boundary conditions. A site can either be empty, or occupied by a vehicle of velocity zero to v_{max}. The velocity is equivalent to the number of sites that a vehicle advances in one update — provided that there are no obstacles ahead. Vehicles move only in one direction. The index i denotes the number of a vehicle, $x(i)$ its position, $v(i)$ its current velocity, $v_d(i)$ its maximum speed, $pred(i)$ the number of the preceding[6] vehicle, $gap(i) := x(pred(i)) - x(i) - 1$ the width of the gap to the predecessor. Note that in the original model all vehicles had the same maximum velocity v_{max}. We now allow for different desired velocities $v_d(i)$ to include an inhomogeneous fleet. At the beginning of each time-step the rules are applied to all vehicles simultaneously (parallel update, in contrast to sequential updates which yield considerably different results). Then the vehicles are advanced according to their new velocities.

- **IF** $v(i) < v_d(i)$ **THEN** $v(i) := v(i) + 1$ (**S1**)
- **IF** $v(i) > gap(i)$ **THEN** $v(i) := gap(i)$ (**S2**)
- **IF** $v(i) > 0$ **AND** $rand < p_d(i)$ **THEN** $v(i) := v(i) - 1$ (**S3**)

S1 represents a constant acceleration until the vehicle has reached its maximum velocity v_d. **S2** ensures that vehicles having predecessors in their way slow down in order not to run into them. In **S3** a random generator is used to decelerate a vehicle with a certain probability modeling erratic driver behavior. The free–flow average velocity is $v_{max} - p_d$ (for $p_d \neq 1$).

Of course, in a realistic setup such as PAMINA the periodic boundary conditions will be replaced by network elements such as intersections (see 2.1.2) or parking accessories (see 2.1.3).

Lane changing

Since a realistic fleet is usually composed of vehicle types having different desired velocities, the single lane model is not capable of modeling realistic traffic behavior. Introducing such different vehicle types in the single lane model only results in *platooning* with slow vehicles being followed by faster ones and the average velocity reduced to the free–flow velocity of the slowest vehicle [6, 23].

We introduce a two-lane model [38, 42] consisting of two parallel single lane models with periodic boundary conditions and four additional rules defining the exchange of vehicles between the lanes. The update step is split into two sub-steps:

[5] Wimmershoff provides an online demo for the single-lane CA model [50].

[6] *A precedes B.* in this context means that A is followed by B

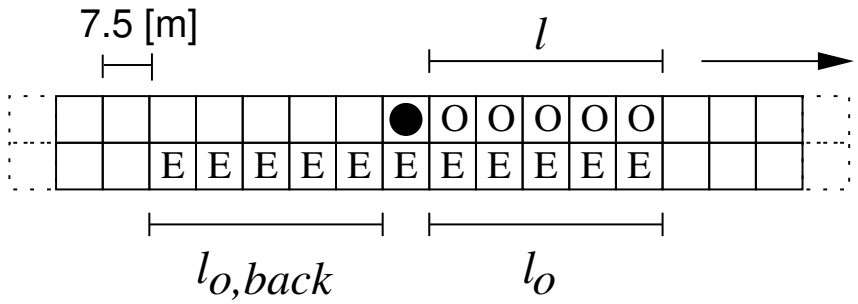

Figure 2.1: *Two-lane CA model: geometry describing lane-changing rules* — The vehicle (denoted by a filled circle) will change to the right (lower) lane if any of the sites marked with "O" is occupied and the sites marked with "E" are empty. The vision ranges are l for look-ahead in the current lane, l_o look-ahead in the other lane, and $l_{o,back}$ for look-back in the other lane.

1. Check the exchange of vehicles between the two lanes according to the new rule set. Vehicles are only moved *sideways*. They do not *advance*. Note that in reality this sub-step regarded by itself is infeasible since vehicles are usually incapable of purely transversal motion. Only together with the second sub-step do our update rules make sense physically.

   This first sub-step is implemented as a strictly parallel update with each vehicle making its decision based upon the configuration at the beginning of the time-step.

2. Perform independent single lane updates on both lanes according to the single lane update rules. In this second sub-step the resulting configuration of the first sub-step is used.

A somewhat generic starting point for modeling passing rules is the following: (T1) The driver looks ahead if somebody is in his way. (T2) The driver looks on the other lane if the situation is better there. (T3) The driver looks back on the other lane if somebody would be obstructed by the lane change.

Technically, we keep using $gap(i)$ for the number of empty sites ahead in the same lane, and we add the definitions of $gap_o(i)$ for the forward gap on the other lane, and $gap_{o,back}$ for the backward gap on the other lane. Note that if there is a vehicle on a neighboring site both return -1. The generic multi-lane model then reads as follows. A vehicle i changes to the other lane if all of the following conditions are fulfilled (see Figure 2.1):

- $gap(i) < l$ (**T1**),
- $gap_o(i) > l_o$ (**T2**),
- $gap_{o,back}(i) > l_{o,back}$ (**T3**), and
- $rand() < p_{change}$ (**T4**).

l, l_o, and $l_{o,back}$ are the parameters which decide how far a driver looks ahead in his own lane, ahead in the other lane, or back in the other lane, respectively.

The most important parameters of the two-lane model are *symmetry*, *stochasticity*, and *direction of causality*. In Table 2.1 we associate the parameters of our rule set with the previously mentioned characteristics.

characteristic	yes	no
symmetry	**T1** for L→R	no **T1** for L→R
stochasticity	$prob_c < 1$	$prob_c = 1$
backward causality	$l_{o,back} > 0$	$l_{o,back} = 0$

Table 2.1: *Characteristics of the two-lane CA rules*

Symmetry: The rule set defining the lane changing of vehicles can be both symmetric and asymmetric. The symmetric model is interesting for theoretical considerations whereas the the asymmetric model is more realistic.

Stochasticity: The single lane model proved that a strictly deterministic model is not realistic. The model did not show the desired spontaneous formation of jams. In the case of the two-lane model the lack of stochasticity in combination with the parallel update results in strange behavior for slow platoons occupying either lane: since none of the vehicles has reached its maximum velocity and all evaluate the other lane to be better there is collective change sidewise which is usually reversed over and over again until the platoon dissolves or the platoon is passed by other vehicles.

We introduce stochasticity into the two-lane rule set to reduce the effective number of lane changes and thus dissolve those platoons.

Direction of Causality: In the single lane model a vehicle only looks ahead (= downstream = in the direction of vehicle flow) so that causality can only travel upstream (= in the direction opposite of vehicle flow). A reasonable lane changing rule must include a check of sites *upstream* in order not to disturb the traffic of the destination lane. This would result in causality traveling downstream.

Active Parameters

In PAMINA III we usually use $l = v+1$, $l_o = l$, $l_{o,back} = v_{max} = 5$, $p_{change} = 1$, and $p_d = 0.5$. Both l and l_o are roughly proportional to the velocity, whereas looking back is not. $l_{o,back}$ depends mostly on the expected velocity of other cars, not on one's own.

In the symmetric version of this model, cars remain in their lane as long as they do not "see" anybody else. If they see somebody ahead in their own lane (i.e. $gap < v+1$), they check the other lane to see if they can switch lanes and do so if possible. Afterwards, if they are satisfied, they remain in this lane until they become dissatisfied again.

In the asymmetric version, cars always try to return to the right lane, independent of their situation on the left lane.

Multilane Behaviour

For each pair of neighbouring lanes the lane changing rules were taken from Section 2.1.1. If a link has more than two lanes, we enforce a left–to–right lane-changing priority. This prevents two vehicles from moving to same common site in case both have to change lanes according to their lane changing rule set.

Speed Limit

In contrast to the original CA model which assumed a maximum speed limit of approximately 120 km/h (freeway traffic), the speed limit within a city is usually lower. In order to match individual speed limits of the simulation area, for each segment we introduced a CA speed limit

$$v_{sl} = \lfloor v_{sl}^{real}/l_{site} + 0.5 + p_d \rfloor$$

where v_{sl} is the CA speed limit given in sites per time-step, v_{sl}^{real} the real speed limit given in meters per second, l_{site} the CA site length given in meters, and p_d the deceleration probability of the CA rule set. Moreover, v_{sl} is forced to be in $[1 \dots v_{max}]$. Note that p_d is used to compensate for its reduction of the average free-flow velocity to $v_{max} - p_d$. Using the speed-limit as a reference for the free speed is based upon the assumption that most drivers will go as fast as permitted. Depending on regional driving habits, the free speeds may have to be adjusted to match actual measurements.

2.1.2 Simple city intersections

In contrast to PAMINA II which features a detailed representation of freeway junctions PAMINA III mainly uses a simple intersection type. For city traffic there is usually no need to provide for transfer lanes since the extent of city intersections can be assumed to be zero[7]. Therefore, the structure can be kept very simple. Figure 2.2 depicts the geometry of a city intersection. All incoming lanes to each segment are equivalent. At the very end of each incoming lane v_{max} sites are scanned for vehicles *before* the usual rules of motion are applied. During each time-step, at most one vehicle per incoming lane of the source segment can be moved to one of the insertion sites of the destination segment. If possible, the same lane is used on the destination segment. If that is not feasible and if the destination segment has fewer outgoing lanes than the incoming segment has incoming lanes, the vehicle is inserted into the leftmost lane. If that site is occupied, the next neighboring site off to the right is checked until a vacant site is found or the right-most lane is reached. Note that the scanning of the incoming lanes is always done beginning at the site nearest to the intersection. Thus, the order of the vehicles with respect to each other is not changed. In order to ensure unbiased processing of all incoming lanes, the scanning is done in a round-robin fashion with respect to consecutive time-steps.

For a vehicle approaching the intersection there are two alternatives: either it is absorbed from the scanning area and inserted into the destination segment or it proceeds according to its CA rules

[7] In TRANSIMS it takes a vehicle at about one time-step to transverse an intersection.

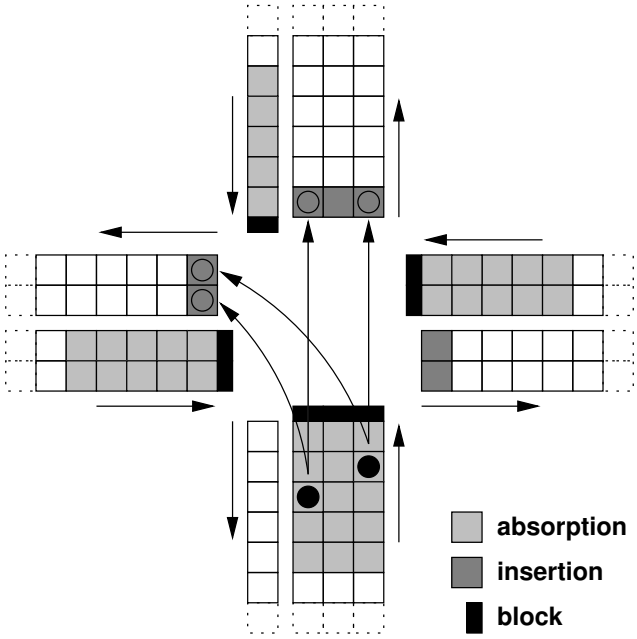

Figure 2.2: *Geometry of a city intersection* — In case of the left turn depicted above, the vehicle on the left lane has no corresponding lane on its destination link. It will be inserted into the leftmost lane.

of motion. Due to blocking at the end of the lane (or earlier due to other preceding travelers) it will eventually stop. This behavior is important to model the spill-backs in real world traffic. The queues are resolved as one would expect: As soon as the situation on the destination lane(s) improves, vehicles are removed one by one starting at the site nearest to the block. Note that a vehicle may be blocked by other vehicles (having a different destination) although its destination segment is vacant. This effect may cause grid-locks which will be discussed in Section 2.2.2.

Approach and turning behavior

In contrast to other traffic simulations with resolution at city-street level, we do not model a special behavior for vehicles while approaching or transversing intersections. In our implementation all incoming lanes are equivalent. This was done for two reasons. First, modeling detailed approach and turning behavior requires extensive geometric information which is often not available or not consistent. Second, the current rules of motion show a quickly decreasing lane-changing probability as soon as the density exceeds a certain threshold. This is mainly due to a strict 'look-back' rule which checks for following traffic on the neighboring lane. In contrast to freeway conditions, where this rule maintains the desired traffic jam waves, here, it would prevent proper lane-changing. This again would result in vehicles queuing up, since they could not change to their respective turning lanes [30].

Traffic lights

Traffic lights are modeled by activating the scanning mechanism for the duration of the green phase T_g and deactivating it for the length of the non-green-phase T_r (which includes both the red phase and transition phases). Since there is only one phase per incoming segment, any direction-specific phasing information is averaged over all directions weighted by the number of active lanes into the respective direction. Let i be the incoming segment, j the outgoing segment, $T_g(i,j)$ the length of the green phase from i to j, and $l(i,j)$ the number of lanes going from i to j. The overall green phase will be computed as

$$T_g(i) = \frac{\sum_j l(i,j) T_g(i,j)}{\sum_j l_g(i,j)}.$$

The overall red phase $T_r(i)$ is computed similarly using all $T_r(i,j)$. Note that due to this averaging, the complete phase cycles $T_{rg}(i) = T_g(i) + T_r(i)$ (green phase + red phase) of the incoming segments may differ from each other, resulting in a continuous phase shift. This is different from real world traffic light installations where the starting time is usually defined by taking a multiple of $T_{rg}(i)$ and adding a relative offset.

Interferences

Two types of interference can occur at an intersection: (a) vehicles that have to obey right of way must wait for gaps in the crossing or oncoming traffic stream, and (b) vehicles that have right of way are obstructed by others blocking the intersection. In the simplest version of our intersection neither of these interferences are handled. However, it is simple to force a reduced throughput through the intersection by examining the overall occupancy of the v_{max} last sites of all outgoing segments and introducing an additional transfer probability. This probability would have a value of one if all sites in the examined area are vacant, a value smaller than one if all are occupied, with a functional (possibly linear) transition between the two extremes.

Sources and sinks

In contrast to a freeway junction, a city intersection may be a vehicle source and/or a vehicle sink. This is required for route-plans which start or terminate at a node. In this case the absorption range at the end of the incoming segments (see Figure 2.2) will be used to absorb vehicles that have reached their final destination. The insertion sites will be used to insert new vehicles.

2.1.3 Parking accessories

Parking accessories are equivalent to sources and sinks, only that they are located right *on* the link. For the simulation of city traffic they serve as insertion (deletion) points for all trips starting (ending) in driveways or parking lots on that link.

Vehicles can be inserted at an accessory if more than 2 consecutive sites are vacant. This is done to guarantee that during heavy congestion not all gaps in the link are filled by vehicles from the accessory source queue.

2.1.4 Route-sets

The first attempt to include route-sets into a medium scale traffic simulation was done in one of the early versions of TRANSIMS [3]: the interstate traffic of the city of Albuquerque was simulated on a single workstation to show the general feasibility of this approach. Nagel [24, 27, 29] used a parallel computer with two CPUs to run a parallel net simulation based upon the single-lane CA with individual route-plans. He examined iterative route-selection behavior for a group of drivers traveling through the network. The NRW-FVU, TRANSIMS, and PARAMICS groups are currently designing or already using large-scale traffic simulations that include route execution. INTEGRATION [20] and DYNASMART [18] have also been used with individual route-plans, albeit at a smaller scale.

For PAMINA, route-sets represent the third major input for the simulation beside nodes and edges. Each route-plan contains information about the node id of the origin, the scheduled departure time-step from the origin, the estimated travel time in simulation time-steps, a list of node id including the destination as its last entry, and optional vehicle data.

PAMINA expects the routes to be sorted according to their departure time-step. Thus the evaluation of the route-plan is reduced to the following scheme: at every time-step routes are sequentially read until a departure time-step is found that is larger or equal the current time step. For each route a vehicle is created and loaded with that route. The vehicle is appended to the insertion queue of the source associated with the given origin id. If there is more than one source per id (e.g. ramps or accessories) also the second node id of the route-plan is scanned to determine the outgoing segment and thus the specific source. Note that the scheduled time of insertion may differ from the actual time of insertion, since the source can only add vehicles to the grid if there are vacant sites. During intervals of high insertion rates there will be a certain number of vehicles waiting (or 'pending') in the source queues.

2.1.5 Levels of fidelity

The principal goal of a CA traffic simulation must be to keep the rule-set as small as possible. This has two advantages: First, by keeping the number of parameters small, the probability of artifacts is reduced and the model can be validated more easily. Second, simple rule-sets usually result in efficient coding which is essential for fast delivery of results — considering the before-mentioned need for statistical averaging.

PAMINA can be run with different model fidelities regarding characteristics of street segments (speed limit, see 2.1.1) and intersections (traffic-lights, see 2.1.2). For each fidelity a specific name is used which is listed in Table 2.2. For the *low-fidelity* (lf) model both traffic lights and speed-limits

were deactivated. In the *speed-limit* (sl) model only traffic lights were activated, in the *traffic-lights* (tl) model only traffic-lights. The *high-fidelity* (hf) model contained both features.

short name	lf	sl	tl	hf	rl
long name	low fidelity	speed limits	traffic lights	high fidelity	reduced lights
speed limit	no	yes	no	yes	yes
traffic lights	no	no	yes	yes	q_r

Table 2.2: *Parameters of simple city simulation* — Parameters (bottom rows) defining the fidelity (right columns) of the simulation: In case of an active speed limit the maximum velocity v_{max} is reduced from its original value of 5 to a segment-specific value. In case of active traffic lights, the transfer at intersections is decreased by introducing periodic red phases during which some of the incoming segments are blocked. Reduced non-green phases (q_r) are discussed in Section 2.2.3.

2.2 Examples

2.2.1 Simulating city traffic using precomputed Routes

As a first test [41] for PAMINA with realistic route-plans, we used a preliminary route-set generated for TRANSIMS case-study. We used two maps (see Figure 2.3): the complete Dallas/Fort Worth area and a small excerpt of the latter called *study-area*. The study-area map comprises all streets except small ones in residential areas and similar areas. The large map further contains all minor and major arterials for Dallas and all major arterials for Forth Worth.

The plan-sets which were available at that time contained only trip departure times between 7 am and 10 am of which we selected those between 7 am and 8 am as the period of interest. Therefore, we started the simulation at 7 am and let it run at least until 8 am. After that, the simulation either terminated when (a) 99% of all route-plans had been executed, or (b) a grid-lock was detected. In this context we assume the system to be grid-locked if the number of vehicles in the system is constant for more than 600 time-steps. For the CA model we used the deceleration probability of $p_d = 0.3$ in all simulations.

For each simulation run all plans can be regarded as static. For the time being, we do not perform any online re-routing. A plan-set is generated from an activity set consisting of a source plus a departure time on the one hand and destination on the other hand. The plan-set used for the study presented here (also referred to as *plan-set 11*) is a very preliminary plan-set which was generated in the course of the Dallas/Fort Worth case-study of TRANSIMS. Work on plan-sets in the context of the same case-study can be found in [21, 26] and in Section 2.4.

All plan-sets are computed for the whole Dallas/Forth-Worth area which means that all routes have to be restricted to the study-area if only that portion of the map is simulated. The truncation of the plans is done in a straight-forward way: any route that contains at least one segment within the study-area will be part of the restricted plan set. Its departure will be delayed by the amount of

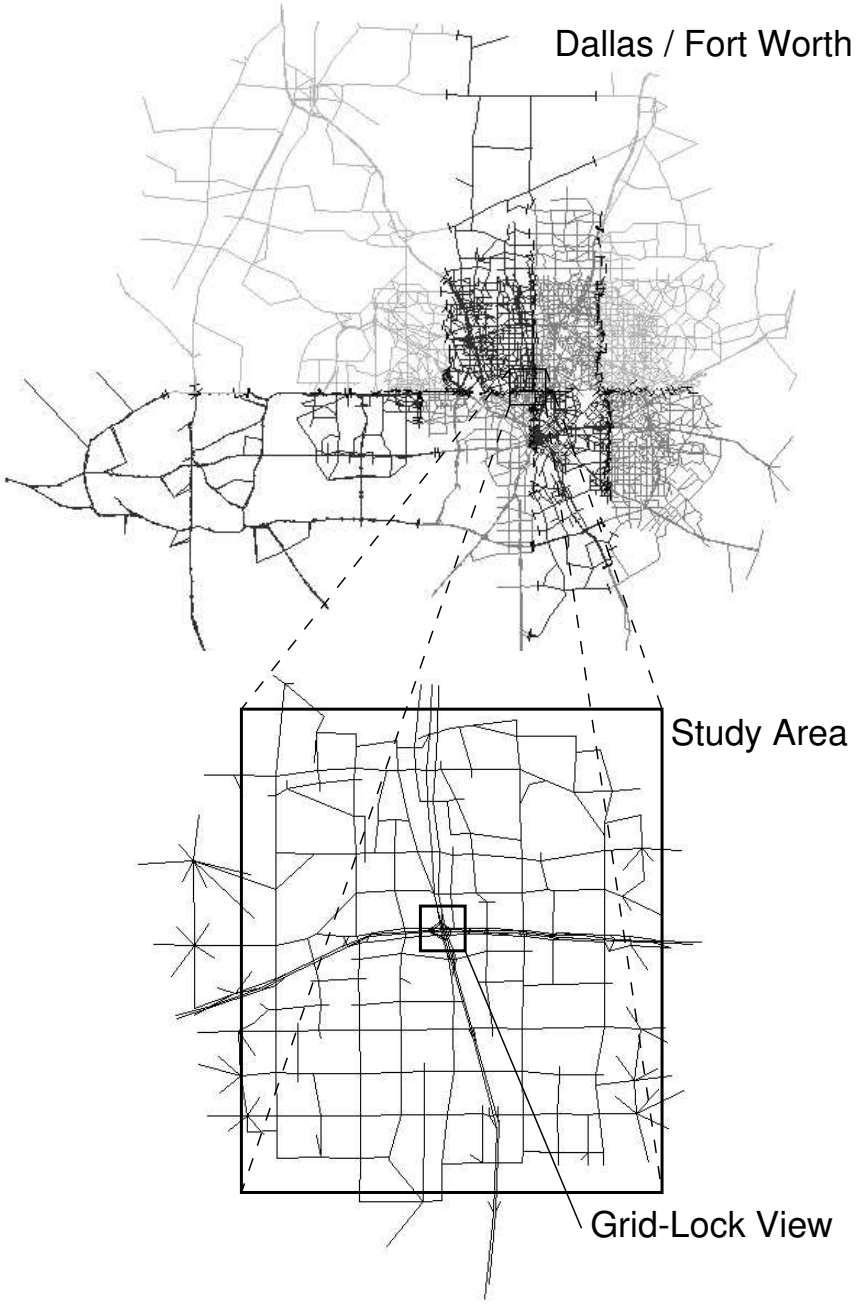

Figure 2.3: *Map of Dallas / Fort Worth* — The different shades of gray in the Dallas / Fort Worth map correspond to the mapping to different processors of the parallel computer topology. Note that the resolution decreases with growing distance from the study-area. For the simulations in this chapter the study-area itself contains only major and minor arterials. The small rectangle marked as *Grid-Lock View* can be seen in Figure 2.8.

time that the vehicle would spend outside the study-area before it reaches the first segment within the simulation area. For all edges transversed up to entry, we use the cruising velocities assumed by the planner (also see 2.5.2).

After the start of the simulation, route-plans are executed as follows: (a) At the time-step given by the departure-time, a vehicle is created at the departure node (source) of the route. (b) The vehicle is inserted into a queue associated with the source. (c) Each time-step the queue is scanned for pending vehicles. If possible, the vehicle is removed from the queue and inserted into the first segment from where it starts executing its route-plan. (d) As soon as it reaches the destination, the vehicle is removed from the segment. The travel time is recorded for statistical evaluation.

Note that all vehicles try to execute their route-plans independent of the actual traffic conditions that they encounter along their way. In heavily congested areas, vehicles often spill back across intersections because they cannot enter their next destination segments. This current approach can result in complete grid-locks of the simulation area, which cannot be resolved with the current rule-set. This artifact will be discussed next.

During the simulation we keep track of: the number of vehicles inserted so far, the number of vehicles currently in the network, and the number of vehicles that have reached their destination. Upon arrival of each vehicle we store the estimated travel time (computed by the router beforehand) and the actual travel time. These times can be compared to check the prediction quality of the router.

Except for the curves depicting the number of vehicles in the study-area (Figure 2.6), we considered only vehicles that arrived before time-step 1800. Also, all curves have been aggregated over 10 simulation-runs (using different random seeds) and normalized according to the respective number of vehicles that have reached their destinations before time-step 1800. Moreover, the area underneath each curve has been normalized to one.

Delay of arrival

Since each vehicle's actual travel time t_{actual} is recorded upon arrival, we compute the distribution of relative delay d

$$d = \frac{t_{act}^{trav} - t_{sched}^{trav}}{t_{sched}^{trav}}$$

with respect to the scheduled trip time t_{sched}^{trav} forecast by the router. Note that negative values denote early arrivals. Figure 2.4 shows the results for plan-set 11 in all fidelities. It is obvious that in mode *lf* (due to the missing speed limit) route-plans are executed much too fast. There are hardly any delays at all. In modes *sl* and *tl* the peak is already shifted towards zero delays but still biased. Mode *hf* generates a distribution which peaks almost exactly at zero delay. The average, however, is shifted towards positive delays. This can be verified in Figure 2.5, which displays the running average (over the latest 1000 vehicles) of the relative delay. In the uncongested regime the travel times of the micro-simulation are shorter in general than what the planner had expected. In highly congested situations, though, travel times in the micro-simulation are longer than the planner predicted.

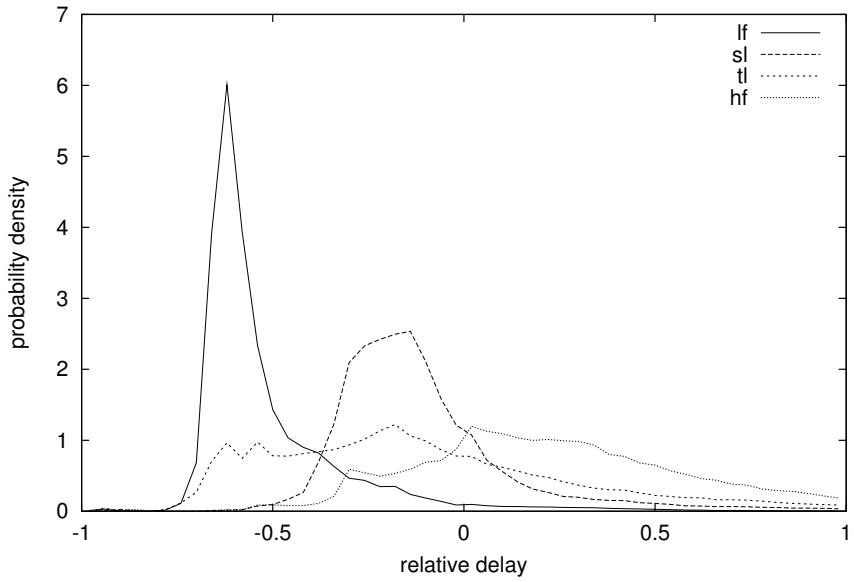

Figure 2.4: *Distribution of relative delays for plan-set 11* — The low fidelity model *lf* shows a characteristics peak at negative relative delays since vehicles are not delayed by either speed-limit or traffic-lights. Higher fidelities have their peaks shifted to positive delays.

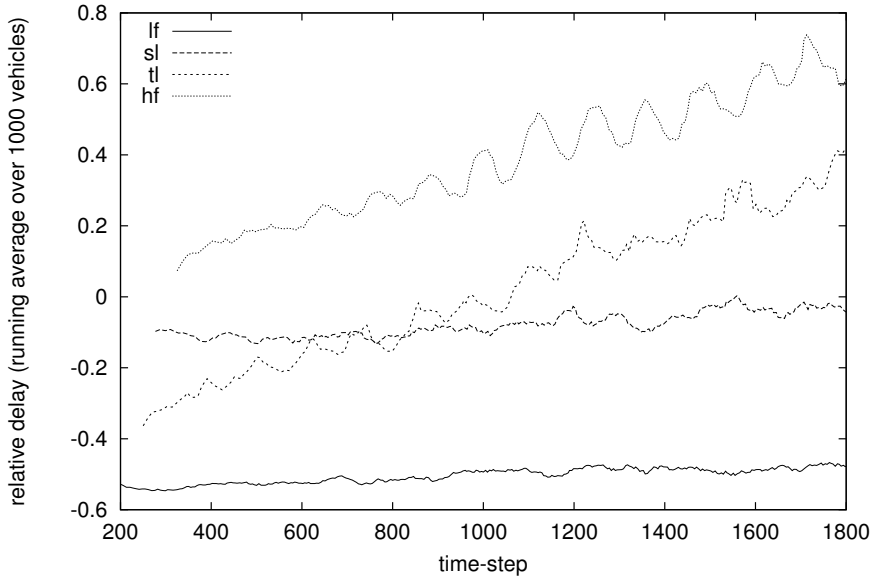

Figure 2.5: *Running average of relative delays for plan-set 11* — The curve for fidelity *sl* is closest to zero relative delay throughout the simulation. Fidelities with activated traffic lights have a rising tendency. This is caused by growing congestion inside the study-area: The number of vehicles never reaches a plateau. The low fidelity simulation executes routes to quickly resulting in large negative delays.

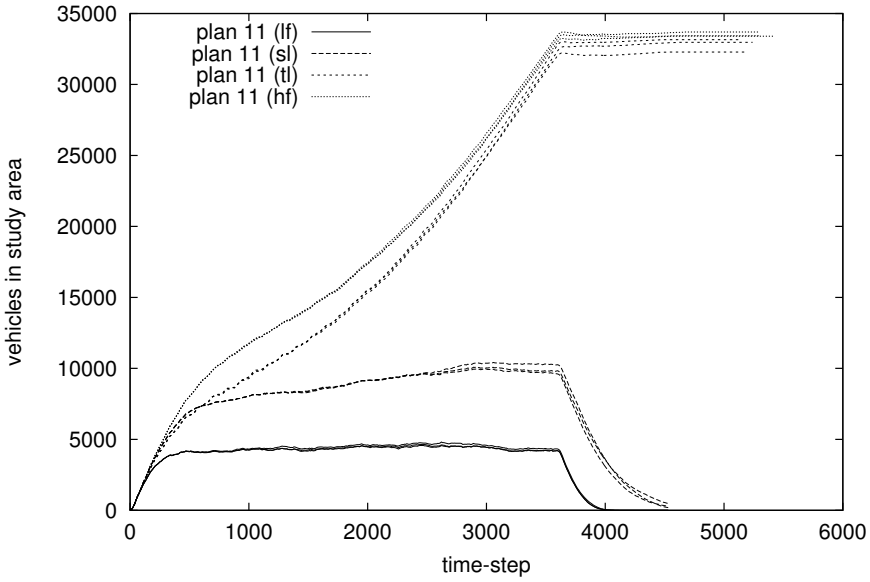

Figure 2.6: *Vehicles in study-area for plan-set 11* — While the vehicle count reaches an equilibrium for fidelities *lf* and *sl*, the other two fidelities grid-lock.

We have to point out that it cannot be the goal of the dynamic micro-simulation to reproduce the static results forecast by the planner. These comparisons only serve as a consistency-check. It is to be expected that results of the micro-simulation will differ considerably, e.g. in places where the implicit aggregation of the planner smoothes over sudden peaks in traffic load.

2.2.2 Reproducibility and grid-locks

Since the CA model contains a stochastic element, we receive a unique evolution of the simulation for each seed of the random number generator. In a sub-critical system the network is able to transport all vehicles (albeit with delay) so that all runs will look similar on a macroscopic level. In a system with a network throughput incapable of handling the loading, the system will most likely grid-lock (see below). Between the two extremes we find a regime in which the specific configuration may either block or not block. Figure 2.6 depicts the number of vehicles which are in the study-area at a given time-step. Fidelities *lf* and *sl* belong to the sub-critical regime. Both curves reach a plateau (at 4500 vehicles after time-step 500 for *lf* and at 10,000 vehicles after time 2500 for *sl*) after an initial loading phase representing an equilibrium between the insertion and deletion rates of vehicles. After the loading phase all remaining vehicles are discharged within 400 time-steps for *lf* and within 900 time-steps for *sl*.

Modes *tl* and *hf* belong to the super-critical regime. They never reach an equilibrium between insertion and deletion during the loading phase; and the plateau after the loading of the network is due to grid-lock.

See [4] for a similar (albeit much smaller scale) investigation on the relation between network

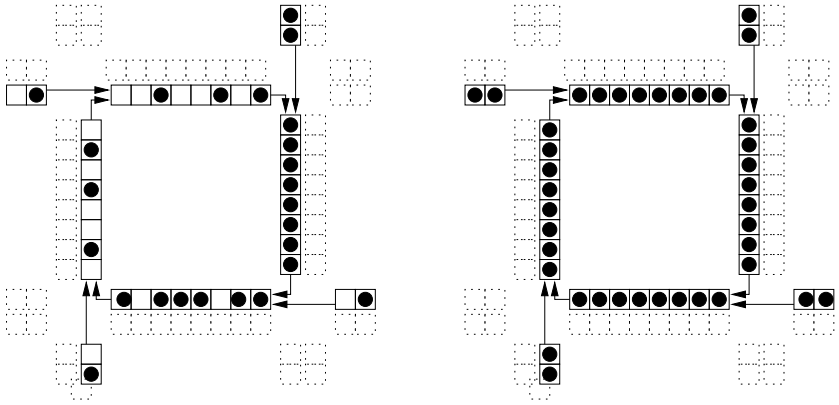

Figure 2.7: *Geometry of a grid-lock* — LEFT: Just before a grid-lock situation. RIGHT: The grid-lock was caused by vehicles which are required to make right turns. The first vehicle of each grid is waiting for the other vehicle to leave the side. Since this dependency is circular, it cannot be resolved anymore.

loading and network throughput.

Grid-locks

In this simulation a grid-lock situation can be determined by a horizontal line after time-step 3600 (e.g. the end of vehicle insertion). This is caused by closed loops in the traffic network in which all sites are occupied. Similar grid lock situations were reported in [13, 26]. Figure 2.7 depicts a simplified intersection. In the left half, traffic is already dense, though not grid-locked. Due to high demand and the red-phases at the intersection, the segments of the loop are no longer cleared. In the right half the whole loop is blocked: the first vehicle in each lane is forced to make a right turn into another lane which is also blocked. This phenomenon (in its strict form) cannot be seen in real-world traffic, because drivers move out of the lanes and pass on the on-coming lane or they abandon their current route and choose a detour. Figure 2.8 shows a screen-shot of a simulation run with plan-set 11 in mode *hf*. Vehicles are represented by dark dots, lane boundaries by grey lines. Right at the center, there is a small grid-locked loop blocking traffic from all incoming directions.

2.2.3 Reduced non-green phase length

We have encountered cases of sub-critical and super-critical loading of the network. Although the respective models are defined by different types of rule-sets, they can be regarded as two specific cases of a more general rule-set (called *rl*) in which in the effective red-phase $T_{r,eff}$ is computed by multiplying the original phase-length T_r by a certain factor $q_r \in [0 \ldots 1]$. Consequently, $q_r = 0$ represents mode *sl* while $q_r = 1$ represents mode *hf*.

We conducted several runs for different values of q_r between 0 and 1. Both Figures 2.9 and 2.10 show a smooth transition between modes *sl* and *hf* (for 0.6 and 0.65 blocking and non-blocking

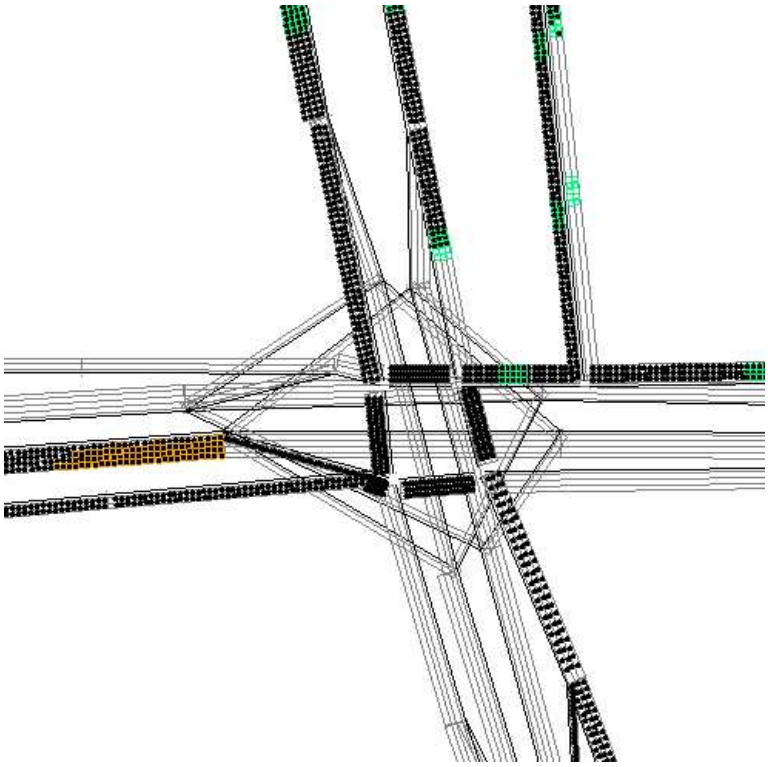

Figure 2.8: *Grid-lock in study-area (plan 11, fidelity* hf*)* — The screen shot shows a grid-lock configuration found while executing plan-set 11 at high fidelity. The location of this excerpt within the study-area can be seen in Figure 2.3.

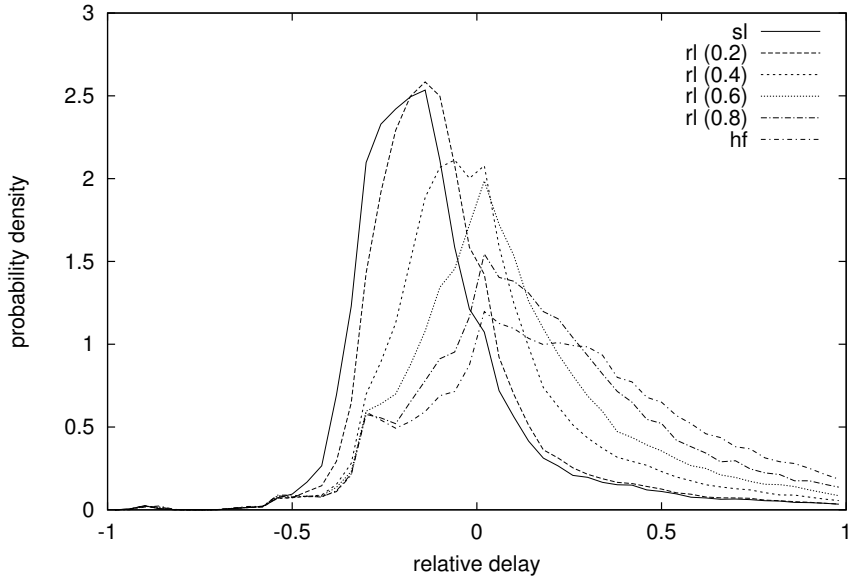

Figure 2.9: *Distribution of relative delay of plan-set 11 (different q_r)* — The distribution of relative delays exhibits a smooth transition between fidelity *sl* (corresponding to $q_r = 0$) and fidelity *hf* (corresponding to $q_r = 1.0$).

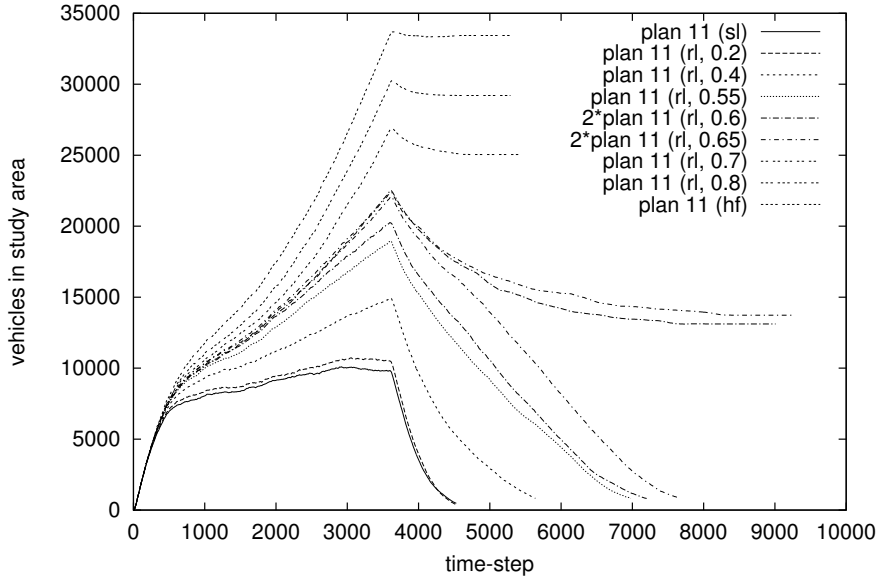

Figure 2.10: *Vehicles in study-area (different q_r)* — The curves for the number of vehicles in the study-area show a transition at $q_r \simeq 0.65$. For larger values the simulation grid-locks, for smaller values it does not.

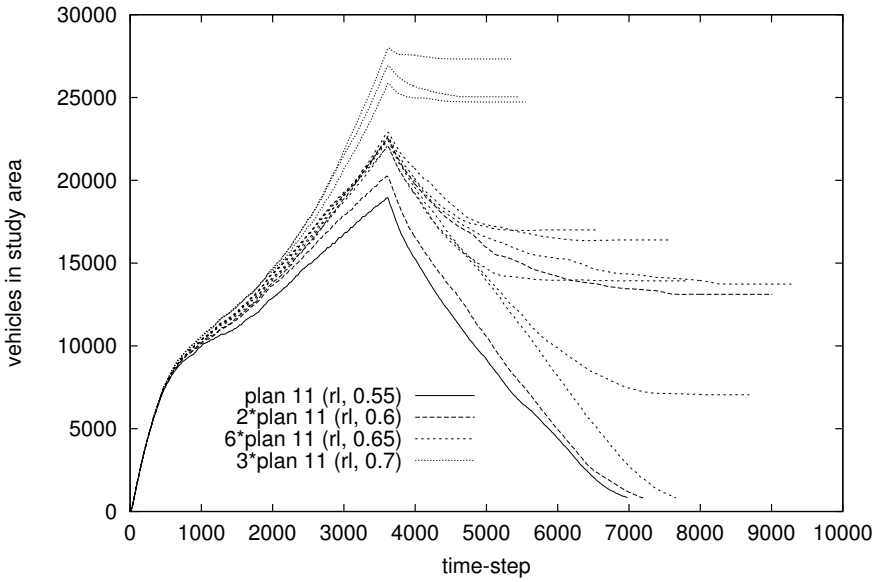

Figure 2.11: *Vehicles in study-area ($q_r = 0.55,\ 0.6,\ 0.65,\ 0.7$)* — For $q_r = 0.6$ and $q_r = 0.65$ the simulation generates both grid-locking and non-grid-locking runs. Note that the only difference between the runs is the seed of the random generator.

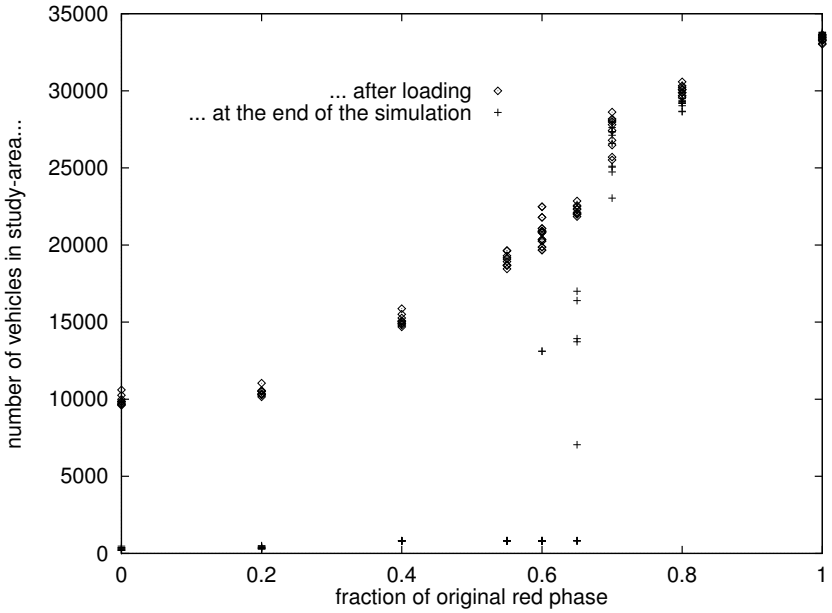

Figure 2.12: *Vehicles in study-area as a function of q_r* — The number of vehicles in the study-area after the loading phase increases almost linear with q_r. The number of vehicles at the end of the simulation phase, however, shows a transition around $q_r = 0.6 \ldots 0.65$.

representatives were chosen) as far as delay and trip duration are concerned. Values below 0.6 show a secure sub-critical (no grid-locks), and values above 0.65 a secure super-critical behavior. For 0.6 and 0.65 the system has a certain chance of reaching a grid-lock (1 out of 10 runs for $q_r = 0.6$ and 5 out of 10 for $q_r = 0.65$), which can better be seen in Figure 2.11. Figure 2.12 depicts the number of vehicles after the loading phase and at the end of simulation as a function of q_r.

A similar effect was reported for simple 2-dimensional grid models [12, 16, 22], except that in these studies the overall density was changed instead of the efficiency of the network components. Intuitively, the grid-lock effect seems to be the same. Further investigations will be necessary to understand in how far simple models on a 2-dimensional grid can indeed offer insight for real-world city traffic, which is happening in 2-dimensional space but is composed of traffic on 1-dimensional links.

2.3 Iterative Route Adaptation

2.3.1 Using origin-destination matrices

Instead of using origin-destination matrices derived from unreliable and incomplete traffic flow counts, another approach was chosen for TRANSIMS [46]. Based on census data[8], artificial households will be created to match the statistical properties of the original census [5]. Each of these households will have a family with members requiring trips (such as home-to-work or work-to-shopping) to different places in the network. In this approach, a combination of trip origin, trip destination, and trip departure time is called an *activity*. The set of all activities for a given time period and area is called an *activity list*. This list can be regarded as a detailed origin-destination matrix that has not been aggregated into time-bins. Note that, for the results presented here, all route-sets relating to TRANSIMS are still based upon origin-destination data, and not on activity lists.

Initial results from the TRANSIMS case-study [26] reveal that not only the route-sets, but also the activity lists, will be influenced by the results of the micro-simulation. This feedback, however, takes place on a different time-scale. Instead of changing route choice on a day-to-day basis, changes to the activity lists rather reflect changes in *when* and in *which order* trips are planned. Drivers facing the same congestion every day will eventually choose a different departure time, combine several trips into one, or change this trip to another day of the week.

2.4 Truly dynamic assignment with simulation feedback

Nagel and Barrett used iterative re-planning for the TRANSIMS case-study [26]. In the remaining portion of this chapter we will concentrate on this iterative approach in which the micro-simulation

[8]This approach may cause difficulties in some European countries where census data is not available for research purposes, even in the strongly simplified form used for TRANSIMS.

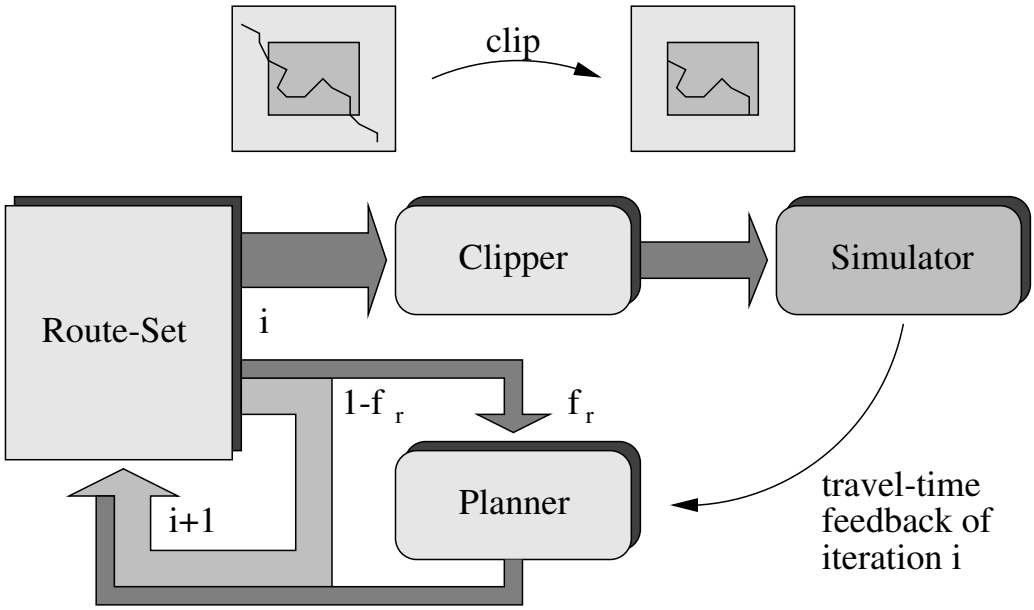

Figure 2.13: *Iterative assignment with simulation feedback* — For iteration $i+1$ the fraction f_r of the previous route-set is re-planned by the router using link travel-times of iteration i. The remaining fraction $1-f_r$ is simply reused. Before the route-set is fed into the simulation it is clipped to the boundaries of the study-area.

and the planner modules are executed in turns to generate a self-consistent route-set. In TRANSIMS, the planner module uses the link travel-times of the previous simulation run to re-plan a certain fraction of the previous route-set (refer to Figure 2.13). This repeated planning process mimics the decision process observed for a fleet of human drivers in which most drivers rely on the routes they used last. The remaining fraction of drivers, however, try to find a new route according to the latest information about the performance of the street network. Since both the demand coded in the origin-destination matrix *and* the travel-time feedback are dynamic, the iterative adaptation with simulation feedback can be regarded as a *truly dynamic assignment*.

2.4.1 Finding the shortest path

The main component of the router is the shortest path algorithm. For a given source, departure time, and destination the router tries to find a route which imposes the least "costs" on the driver. The usual approach is to define time-dependent link costs $c_j^{trav}(i)$ (also called *weights*) for each link j and each time-bin i. As mentioned before, costs can be a combination of travel-time, distance, and financial costs (such as tolls). They may also include preferences for certain street types (such as scenic roads) or prohibitions (trucks through residential areas). Therefore, a better expression for the shortest route would be the *minimum cost route*.

In principle, each driver has an individual *subjective view* of the costs of a link depending on his financial and social background. It is possible that two drivers choose different routes because their link costs vary considerably, and neither of the routes has to be the shortest route (measured

as the sum of the Euclidean lengths of the links) or fastest route (measured as the sum of travel times on the links). For the work presented here, however, we regard the whole fleet of drivers as homogeneous and use the *link travel time* as the only cost factor. This may cause certain instabilities because minor changes in link travel-times may result in many drivers modifying their routes collectively. Nagel [26] reports a positive effect when a 30%-noise factor was added to the original cost function to create an artificial individual view of the network.

Once the link costs are defined, there is a wide variety of algorithms available to compute an exact solution of the minimum cost route. Among those, Dijkstra's algorithm (see i.e. [2]) is probably the most commonly used. Variations of the original algorithm try to improve the run-time performance depending on specific characteristics of the cost functions and the underlying graph (see [10]). In the case of a street network which is basically planar and the number of edges E is approximately as large as the number of nodes N (except for a constant factor), Dijkstra's algorithm has a complexity of $O(N \log N)$ for the computation of all minimum cost paths from one source to all other destinations in the network. This version will be used in this chapter within the router module.

2.5 Using PAMINA for truly dynamic assignment

At this point we will use the micro-simulation PAMINA to iteratively adapt a route-set consisting of all routes going through an 8x8 [km] area inside Dallas. The map includes all street types such as highways, arterials, and local streets in residential areas. For the case-study experiments, the initial route-set was derived from trip-table provided by the North Central Texas Council of Governments (NCTCOG). Refer to [26] for a more specific description of this data.

2.5.1 Router and feedback data

In this section we outline the router used in the case-study and the nature of the data used as feedback from the micro-simulation to the planner.

During the micro-simulation each link of the street network is constantly monitored. Every 10 time-steps the current velocities of all vehicles on each link are accumulated. Every $t_b = 900$ time-steps (15 minutes) the values are converted to link travel-times and written to a file. During grid-lock on a link the sum of velocities is zero which would result in an infinite travel-time. Therefore, the travel-time is limited to the time a vehicle would spend on the link if it were going at 1/100th of the free speed of the link. If the link is vacant, the travel-time is set to be exactly free speed.

The TRANSIMS iteration uses two files for its feedback process. The first file, equivalent to the one described above, contains average link travel-times. The data, however, is retrieved differently: vehicles leaving links trigger events which are collected in a statistics file. After the simulation a post processor aggregates all travel-times into time-bins. There are two reasons for a vehicle count of zero: (a) no vehicle left the link, because the there was no traffic on the link, or (b) no vehicle

could leave the link because a link up-stream was completely grid-locked. A second statistics file containing the link occupancies is used to decide which case applies.

The planner reads the link travel-times $t(i)$ and assigns them to 15-minute time-bins indexed by i. For each route of the input route-set (e.g. the route-set of the previous iteration) the planner determines whether it is re-planned or simply written to the output route-set without any modification.

When the route is to be re-planned, the planner applies Dijkstra's shortest-path algorithm using the previously collected link travel-times. Two slightly different versions of the planner (RP1 and RP2) are used. For both of these versions, T is the distance of the node which is to be labelled next in Dijkstra's algorithm. It corresponds to the wall-clock time at which a vehicle following the shortest path would arrive at the given node.

RP1 is the route-planner which was also used in the iterative reference run of the TRANSIMS project. It uses the time-bin $i = \lfloor (T + t_b)/t_b \rfloor$ which corresponds to a "look-ahead" time-shift of t_b. This improves the relaxation of the TRANSIMS iteration by anticipating congestion on links [26]. Wunderlich et al [51] investigated the impact of delay of travel time feedback on the convergence of the iterative routing process. They also report an improvement with increasing "back-dating" of travel time information.

For time-bins after 12:00 pm the planner assumes free speeds for all links again. Turning prohibitions at intersections are handled by preventing the shortest path algorithm from proceeding into any prohibited direction. Note that this approach may exclude complicated paths that visit a node more than once to circumvent prohibitions. Since the number of affected plans is very small, this error can be ignored.

RP2 is the research version of the planner. Since its source code is more easily accessible it is used for most of the runs. In contrast to RP1, the time-bin is computed as $i = \lfloor T/t_b \rfloor$. Also, in contrast to RP1, the research version completely ignores left turn prohibitions. For time-bins after 12:00 pm, the travel-times of time-bin 11:45 am are used.

In both cases, $t(i)$ can only be regarded as an estimate, since it is averaged over a period of 15 minutes. RP1 tends to take time-bins which may be too far in the future, whereas RP2 does the opposite: it only uses data which is slightly outdated. A more exact approach would be to use T to compute the first bin i as RP1 does, but use later time-bins if the travel-time on the link is longer than the bin width. This way, we average the travel-time over all bins that the vehicle "visited" while it was on the link. The first and the last bins are only weighted with the respective fraction that they were used.

2.5.2 Route-set conversion

Before the route-set can be used as input for the simulation, the routes extending over the whole planning area of Dallas - Fort Worth have to be restricted to the simulation area as follows. For

each route of the original route plan, determine the first link inside the study-area. If the estimate of the entrance time into that link lies within the simulation-time window (5:00 am to noon), the beginning of the route is restricted to the first link inside the area. The entrance time into the study-area is used as the new departure time. If the route leaves the study-area again, the remaining portion of the route will also be removed. The new scheduled travel-time corresponds to the time spent inside the study-area. After conversion, all routes are sorted according to their new departure times.

Also note that, in principal, there may be routes entering and leaving the simulation area more than once. The current version of the micro-simulation treats these routes as though they had only one entrance point into the study-area. The fraction of affected routes can be assumed to be small due to the convex shape of the study-area.

2.5.3 Iteration parameters

The re-planning process as it is used within the current context is defined by a small set of parameters: the choice of the *initial route-set*, the *re-planning fraction*, and the *route selection*.

Initial route-set

At the start of the iteration there is a choice between three different initial route-sets that were generated from the O-D matrix based on

- free speeds (called *FS*) as time-independent link-weights,
- the logical link-lengths (called *SP* for shortest path) as time-independent link-weights,
- an empty route-set (called *VD* for void) in which all routes were deactivated. In this case "re-planning" a route means planning the route for the first time and activating it. Once a route is activated it remains activated throughout the iteration.

Re-planning fraction

Choosing the re-planning fraction f_r is based on a trade-off between computational speed and stability of the iterative process. Large fractions allow fast re-planning of all routes in the initial route-set. As we will see later on, one prerequisite of relaxation is that most of the routes have been re-planned at least once. As the iteration proceeds, however, large fractions have the clear disadvantage of moving routes back and forth between similar alternatives (see [26]).

The aim of the iteration process should be either to keep the re-planning fraction small throughout the process or at least reduce the fraction more and more as the iteration approaches relaxes.

Route selection and route aging

The above-mentioned re-planning fraction does not yet define *which subset* of the previous route-set is to be re-routed. Although there are a multitude of different approaches of how to select routes, for simplicity we have decided to concentrate on using the age (denoted as "a") of a route (the number of elapsed iterations since it was last re-planned) as the only parameter. In particular, we chose the following versions of route selection:

random The routes are selected at random with the probability f_r. The planner does not distinguish between routes in any way. All routes are picked with the same probability $P_{rnd} = f_r$.

scheduled random The routes are also randomly selected, but the re-planning fraction was explicitly chosen for every iteration. We used this scheme only for the first two test-runs, since it required user interaction. The iteration of the TRANSIMS-14 run was also based upon a schedule.

linear age The likelihood P to select a route of age a is given by $P_{lin}(a) = qa$ were q is a factor that only depends on f_r and is thus a constant of the iteration.

forced reduction with random smoothing The N routes are split into two groups: those that have never been re-planned, and those that have been re-planned at least once. For n iterations we re-plan N/n plans of the first group and an additional fraction of f_r of the second group. For iterations after n, only the second portion of re-planning remains. The probability in iteration i is computed as $P_{red} = f_r$ if $i > n$ or the route has already been re-planned, and $P_{red} = 1/(n + 1 - i)$ otherwise.

All of the above versions eventually lead to a stationary age distribution $f(a)$ which can be analytically predicted. The first and the last versions result in a decreasing exponential distribution with

$$f_{rnd}(a) = f_r(1 - f_r)^a,$$

the second one results in a normal distribution

$$f_{lin}(a) = f_r e^{ba^2}.$$

Since the linear age version tries to re-plan older routes sooner than younger ones, the age distribution is biased towards younger plans compared to the random version.

2.5.4 Relaxation

So far we have only described how we execute the iterative process and what parameters we use to influence it. A very important aspect is how to measure the actual improvement achieved by re-planning. As we have seen in Section 2.2.1 the micro-simulation exhibits grid-lock whenever loops in the network links are completely filled with vehicles. One obvious improvement would

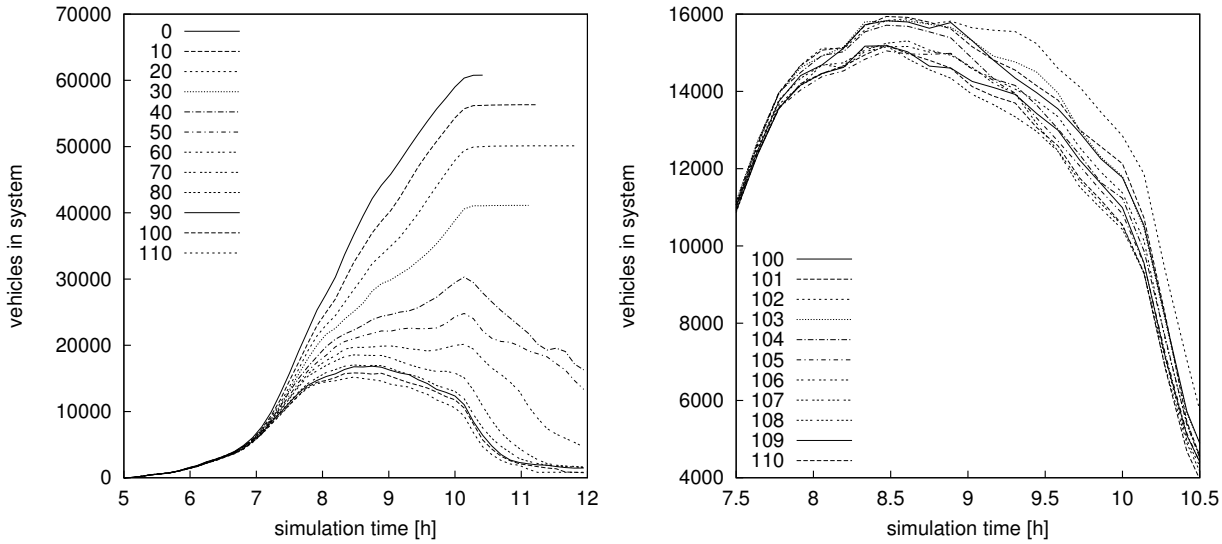

Figure 2.14: *Run 4: Number of vehicles in the study-area* — LEFT: For early iterations the number of vehicles in the study-area exhibits grid-locks (horizontal lines). As the process proceeds the grid-locks disappear until eventually most of the vehicles are able to leave the study-area. RIGHT: The last ten iterations show very little variation. Note the different scale! The number of vehicles in the study-area proves to be an impractical measure for the progress of the iteration process.

be a reduction of the number of grid-locks. The left-hand side of Figure 2.14 shows the results from an iteration using random route-selection and a re-planning-fraction of $f_r = 0.01$. We see the number of vehicles in the study-area plotted against the simulation time (one curve for every tenth iteration). It is easy to detect two transitions in the figure. First, between iterations 30 and 40 the simulation stops grid-locking with the vehicle count remaining constant after a certain point in time. Second, between iterations 60 and 70 the throughput of the system has improved so that practically all vehicles are able to leave the study-area during the simulated time. As the iteration continues and all grid-locks have been dissolved, these curves get less and less informative since the absolute difference between simulations decreases. The right-hand side of Figure 2.14 shows the last 11 of the 110 iterations. Although there is still some improvement visible between the set of 100 through 104 and the set of 105 through 110, the change is not as drastic as in earlier iterations.

Changing the display from the number of vehicles in the study-area to the aggregated travel-time gives more insight. The left-hand side of Figure 2.15 shows a continuous decrease with some underlying noise. The improvement is mainly caused by three factors:

a) The traffic volume is increasingly distributed among all street types in contrast to the original route-set which is biased towards fast routes (FS) using the network hierarchies in a heterogeneous fashion.

b) Since the capacities of the access links to the study-area are limited, there is a growing spill-back of vehicles at the boundaries as soon as demand exceeds capacity. The feedback for

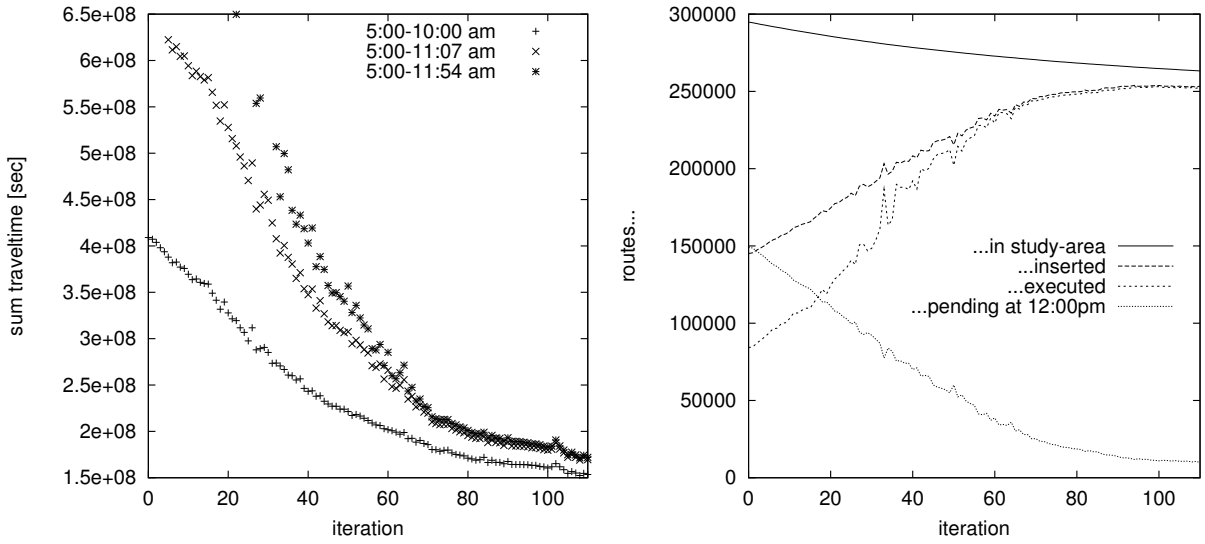

Figure 2.15: *Run 4: Sum of travel-times and executed routes* — LEFT: Accumulated travel-time tt_{acc} of all vehicles in the study-area as a function of the iteration number. It is obvious that despite the high number of iterations, the value of tt_{acc} has not sufficiently relaxed. RIGHT: Number of routes routed through the study-area, those inserted into the study-area, and finally those successfully executed by noon.

link travel-times does not punish those queues in the original version. In Section **??** we will introduce a correction to the feedback which takes care of queue delays.

c) The number of routes that are routed through the study-area decreases. The right-hand side of Figure 2.15 depicts the number of routes planned, inserted, and executed. After 110 iterations only 263281 of the initial 294883 routes remain reducing the traffic load by roughly 10%. This artefact will also be discussed in Section **??**.

run	4	5	7	8	10	11	12	13/16	14	15	17
planner	RP1	RP2	RP2	RP2	RP2	RP2	RP2	RP2	RP2	RP2	RP2
init. route-set	FS	FS	VD	SP	FS	FS	FS	FS	FS	FS	FS
iterations	110	110	60	60	20	60	60	60	80	80	60
reductions	-	-	-	-	20	-	-	-	-	-	-
fraction f_r	0.01	0.01	0.05	0.05	0.01	0.05	0.05	0.05	0.05	0.05	0.05
selection	rnd	rnd	age	age	red.	age	rnd	age	age	age	rnd
level-0-corr.	-	-	-	-	-	-	-	-	lin.	sqrt	-
queue feedb.	-	-	-	-	-	-	-	yes	-	-	yes

Table 2.3: *Parameter combinations of iteration runs* — For the initial route-set *FS* denotes *free speed*, *VD* denotes *void* and *SP* denotes *shortest path*.

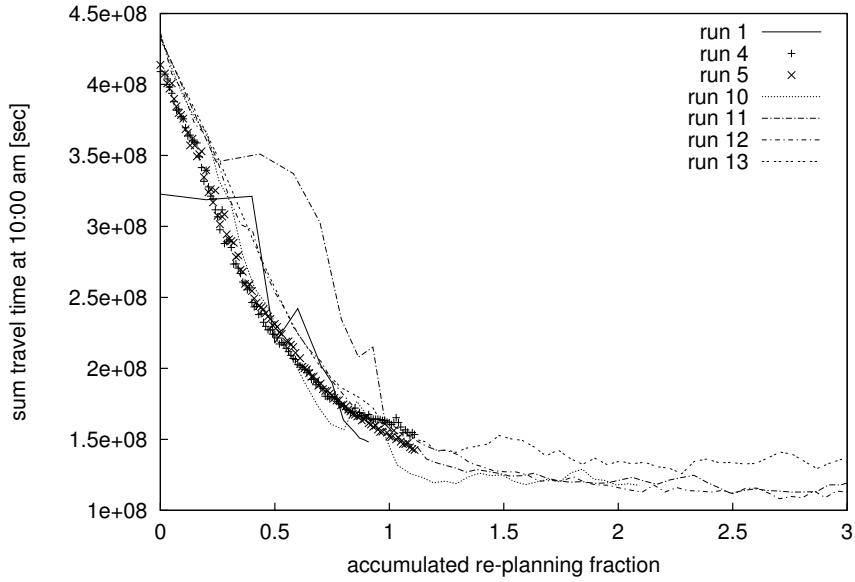

Figure 2.16: *Relaxation by accumulated re-planning fraction (all iterations)* — With respect to the accumulated re-planning fraction all curves (except for run 13) collapse into one. The differences of runs 1 and 11 are caused by exceptionally few or many grid-locks, respectively. Also see Figure 2.17.

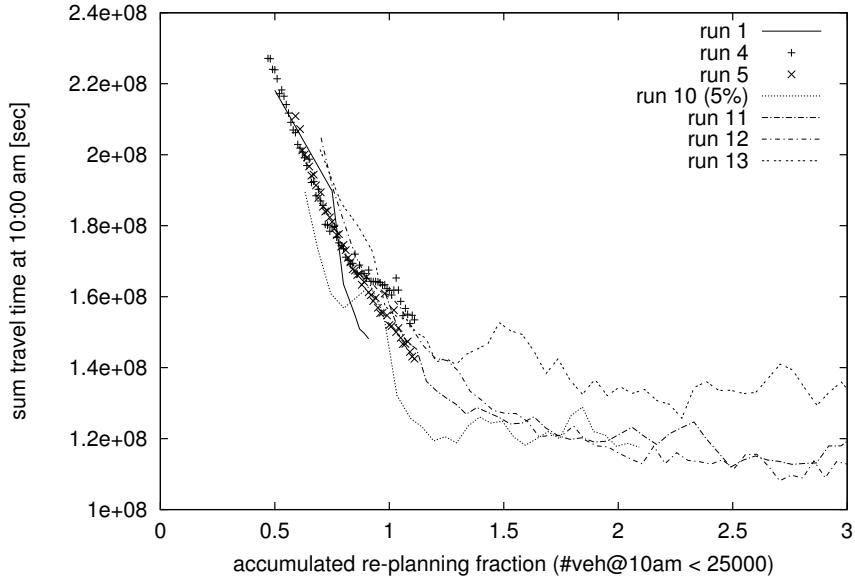

Figure 2.17: *Relaxation by accumulated re-planning fraction (non-grid-locking iterations)* — The curves were restricted to those iterations with less than 25,000 vehicles in the study-area at 10:00 am.

Accumulated re-planning fraction

A very useful comparison of the replanning success can be obtained by using the *accumulated re-planning fraction* f_{acc} as the ordinate of the plot. This fraction is defined as the sum of all individual

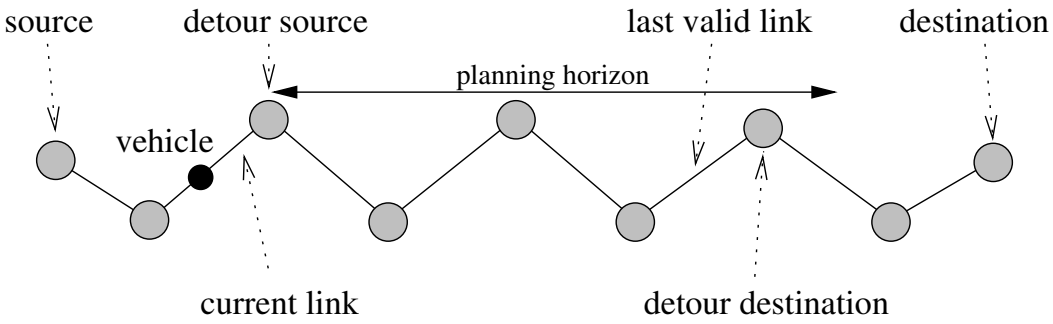

Figure 2.18: *Geometry of an online-detour* — For a vehicle the head of its current link serves as the source node for the detour search. The last node that is still within the planning horizon serves as detour destination. See table 5.19.

re-planning fractions up to a given iteration. Figure 2.16 shows the same curves as in Figure **??** plotted against f_{acc}. The curves for all runs coincide very well. Runs 1 and 11 exhibit some points outside the general slope. This is due to extreme grid-locking of run 11 during early iterations and very few grid-locks during the first iteration of run 1. Note that run 13 levels off at a slightly higher sum travel-time than the other runs. This will be discussed in **??**.

In Figure 2.17 the similarity becomes even more obvious after removal of all iterations with vehicle counts below 25,000 at 10:00 am. The peaks of runs 1 and 11 have disappeared. Run 10 is the first one to reach a sum travel-time of approximately $1.2 * 10^8$ shortly after an accumulated re-planning fraction of one. At this point, it is the only run in which *all* routes have been re-routed at least once.

2.6 Online Routing

2.6.1 Re-routing algorithm

So far we have used static route-sets for all simulations with PAMINA. Each vehicle followed its route-plan independent of the *current* road conditions until it reached its destination. This approach, in conjunction with the discreteness of the traffic model, resulted in grid-locks which could not be resolved (see Section 2.2.2). Using iterative re-planning, grid-lock no longer occurred in later route-sets.

The next question is, how to enhance the static approach by using an online re-planning scheme. "Online" in this context does not mean that we feed real-world online state information (e.g. data provided by online monitoring devices such as counting loops or video cameras) into the re-planner. Instead, we use the micro-simulation to provide state information. The starting point for each experiment is a route-set obtained from the iterative re-planning process. This serves as a specific "test day" for the traffic in the simulation area. Without online re-planning we test the quality of this route-set by executing the route-set several times using different random seeds. For each route through the study-area we obtain average trip-times and their respective variances.

2.6.2 Criteria triggering re-routing

Using the static case as the base-case, we now allow a certain subset of drivers to access online information during the trip through the study-area. The fraction of drivers equipped with these intelligent routing devices is called the *market saturation* m_{o-l}. Every t_{o-l}^{update} time-step (in the range of $120 \ldots 240$ seconds) all equipped vehicles are monitored. For each vehicle the following steps are executed (see Figure 2.18):

- Starting with the *current link* (ending in the potential *detour source node*), the vehicle adds up the current travel-times t_{o-l}^{trav} for all future links in its route-plan until it reaches the first link whose head is further[9] away from the detour source node than the given planning horizon h_{o-l}. For all remaining links the travel-times contained in the original route are used. The sum of all individual travel-times yields a new arrival time T_{o-l}^{arr}.

  Note that, here, the "historic" information of the old route is used to allow some reasonable comparison. The situation that we are investigating can be compared to the daily commute to work for which each driver memorizes his previous trip time. Of course, there are other ways to include older information. In a day-to-day simulation of the street network of Nordrhein-Westfalen (NRW) [25], however, it was shown that there is no significant difference between using only the previous trip or additional older trips (in this case with an exponentially decreasing weight).

- The driver compares his scheduled arrival time T_{sched}^{arr} contained in its original route-plan to the new estimate T_{o-l}^{arr} provided by the online router. If the relative estimated delay

$$d_{o-l} = \frac{T_{o-l}^{arr} - T_{sched}^{arr}}{t_{sched}^{trav}}$$

  is greater than a given threshold d_{o-l}^{min} the driver will *request* an alternative route from the online router. Otherwise (and in the case that the vehicle is only two links away from the destination) no action is taken.

- The online router starts a shortest-path search using the *detour source node* and the to-node of the last valid link as *detour destination node*, which is the last node of the current route within the planning horizon. The router computes a new estimated travel-time for the re-routed portion which is combined with the links from the detour destination node to the ultimate route destination node, yielding a re-routed arrival time T_{r-r}^{arr}. The relative re-routing delay (negative values denote an improvement) is given as

$$d_{r-r} = \frac{T_{r-r}^{arr} - T_{o-l}^{arr}}{t_{sched}^{trav}}.$$

- If d_{r-r} is smaller than a given threshold d_{r-r}^{min} the vehicle uses the new alternative route. This is counted as a re-routing event. The router will increment a counter associated with the

[9] Manhattan-distance

vehicle to keep track of how may times the vehicle has been re-routed. The estimates for all links up to the detour destination node are replaced by those of the new route. All estimates after that (including the arrival time) are corrected by adding the time difference $T^{arr}_{r-r} - T^{arr}_{sched}$ to the old values.

2.6.3 Shortest-path algorithm and edge weights

During the simulation, each link j (with length L_j, number of sites S_j, and free speed v_j^{free}) is monitored every t_{sample} time-steps for vehicles. The number N_j of vehicles found on link j and the sum of their velocities V_j^{sum} are accumulated over the update interval of t_{o-l}^{update} time-steps. At the end of each interval the travel-time t_i^{trav} is computed as follows:

$$t_j^{trav} = \begin{cases} N_j L_j / V_j^{sum} & \text{if} \quad V_j^{sum} > 0 \\ L_j / v_{min} & \text{if} \quad V_j = 0 \quad \text{and} \quad N_j / S_j > \varrho_{thresh} \\ L_j / v_j^{free} & \text{otherwise.} \end{cases}$$

For a given source S, destination D, and departure time T^{depart} the router executes the following steps: If S is the same as the previous source S_{prev} and $T^{depart} = T^{depart}_{prev}$ we re-use the shortest-path tree of the previous computation. Otherwise, we compute a new shortest-path tree using a simple label-setting Dijkstra [10] algorithm. The travel-times t_i^{trav} are used as link weights. Only those nodes that are in a Manhattan distance of h_{o-l} to S are considered for labeling. The algorithm stops only after all reachable nodes have been labeled, even if D has already been labeled. This allows the router to re-use the shortest path tree several times, since the requests are processed link by link. All vehicles on the same link share at least the first node (source) of their potential detour. The label at D is used as the travel-time from S to D.

Note that this algorithm works with time-independent edge-weights. Within the context of the test-bed this can well be justified, since the average travel-time of all routes is about $10 \ldots 12$ minutes. For re-routing in larger areas, one would have to include travel-times obtained from sources other than current measurements. In this case it is still possible to compute the shortest paths in an efficient manner on a parallel computer (see e.g. [11, 44]).

2.6.4 Re-routing parameters

In Section 2.6.2 we have described four parameters that will be used to influence the behavior of the online re-planning. These are:

Market Saturation The impact of the market saturation m will be important for the marketability of online re-routing devices. Potential customers will only accept devices that provide a certain guarantee of success. If fees for route alternatives are only due when they were successful (e.g. resulting in a travel-time shorter than some predefined average) providers will be interested in estimating what minimum success-rate would be necessary to break even.

Planning Horizon The planning horizon h_{o-l} (maximum spatial look-ahead) of the detour search has an important impact on the run-time behavior of the shortest-path algorithm. In a traffic map, intersections are homogeneously distributed over the map area. The number of nodes to be labeled within a given distance h_{o-l} from the source node scales with $O(h_{o-l}^2)$. For this reason, the value should be kept as small as possible. On the other hand, increasing the value may enable the algorithm to find more useful alternatives.

Note that a small h_{o-l} not only reduces the number of labeled nodes considerably, it also decouples re-planning regions in a distributed route guidance system (see e.g. [44]).

Update Interval The update interval for the link travel-times T_{o-l}^{update} will represent a trade-off between a quick response to increasing demand on links (short intervals), on the one hand, and good link travel-times statistics on the other hand. As far as computational efficiency is concerned, shorter intervals also increase the amount of communication between re-planning regions. This is directly transferable to the hardware requirements of a real-world route guidance system: the faster the update, the higher the required bandwidth.

Accumulated Re-planning Fraction The result of the iterative re-planning process is a self-consistent route-set which exhibits little fluctuation with respect to additional re-planning. The question is if reality is actually that close to a possible equilibrium. In case it is not, route-sets obtained from earlier iterations may be more similar to a real-world configuration of route-choices. Therefore, we will use route-sets from different iteration runs as base-cases for our re-planning experiments.

2.6.5 Simulation setup

For each experiment we run the simulation five times with no online re-planning ($m = 0$) to obtain data for the base-case. Each run uses a different set of random seeds. Afterwards we run a set of five runs for each parameter combination that we investigate.

In contrast to the iteration runs presented in Chapter **??** we changed the behavior of the planner RP2 slightly. In order to provide current travel-time estimates for all drivers, all routes, even those that are not re-planned, are updated to reflect the travel-times of the previous simulation run. Otherwise, the incentive to request a new route-plan would be considerably reduced for early iterations, since most of the drivers still have estimates dating back to iterations when grid-locks were very common. Figure 2.19 shows relative delays for runs 13 and 17 which use the same parameter combination but differ in the travel-time estimates. Run 17 shows time-independent behavior whereas run 13 starts out by underestimating travel-times but considerably overestimates travel-times later in the simulation. We chose the parameter combination

- planning horizon $h = 5$ $[km]$ (corresponding to half the diameter of the study-area),
- update interval $T_{o-l}^{update} = 120$ $[sec]$, and $t_{sample} = 10$ $[sec]$,
- accumulated re-planning fraction $f_{acc} = 1.0$, and

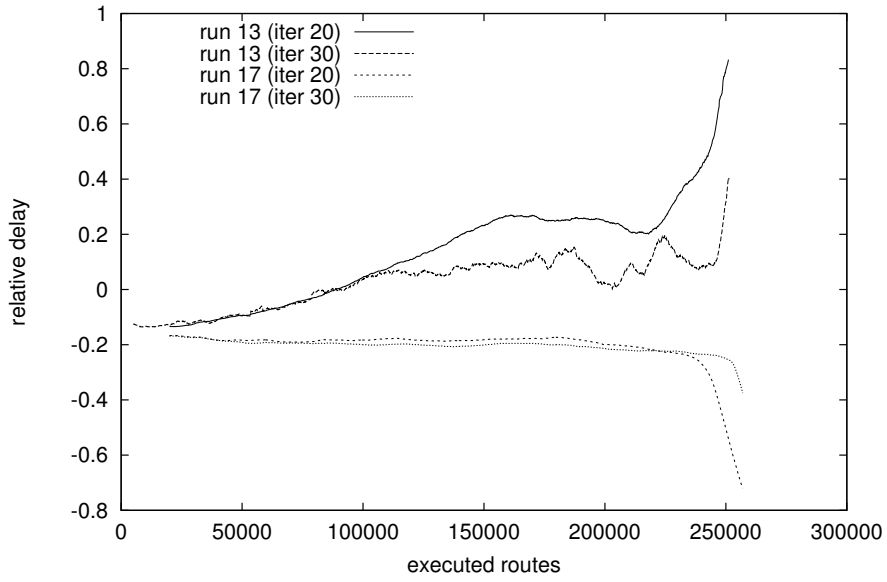

Figure 2.19: *Update of travel-times* — The curves show the running average of the relative delay for different runs. While run 13 uses scheduled arrival times based upon old link travel-time feedback files, run 17 re-computes all estimates using the latest run.

- $v_j^{min} = 0.01 * v_j^{free}$,
- $d_{o-l}^{min} = d_{r-r}^{min} = 0.1$, and $\varrho_{thresh} = 0.05$

to be the reference for all our comparisons. These and most other combinations were sampled in intervals of 10%, for market saturations between 0 and 90%.

2.6.6 Recurrent congestion

The first case that we investigate is that of recurrent congestion. The route-set that is used for the simulation contains a configuration of drivers that have chosen their routes over a re-planning period of 20 iterations. Each driver has chosen his route based on a snap-shot of the previous iteration. Since the micro-simulation is non-deterministic, the traffic conditions in areas with large variance may be completely different from the current run. Therefore, there is a certain chance that choosing a different route will improve the trip-time of a driver although, *on the average*, the current route is already a good choice. Online routing comes into play when the current snap-shot is used instead of an outdated one.

Quality of service

When subscribers are offered a route alternative they expect an improvement in travel-time relative to the original route they would have taken. Unfortunately, due to the uncertainty of state infor-

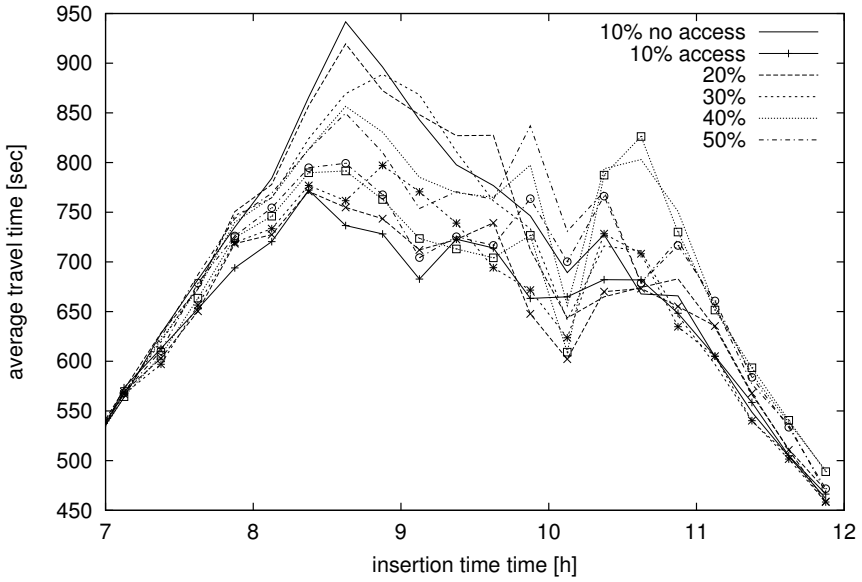

Figure 2.20: *Quality of service (iteration 20)* — Average travel-times of non-subscribers (lines without symbols) and subscribers (corresponding lines with symbols).

mation and prediction, the alternative may not live up to its promise. Therefore, it is important to have some quantitative measure about how well the re-routing process works.

First we will look at the overall benefits for subscribers compared to non-subscribers. We compute the average travel-time of all subscribers and non-subscribers and average them over time-bins (with respect to their insertion time) of 15 minutes. In Figure 2.20 the average travel-time is plotted for different market saturations. Prior to 10:30 am the curves for non-subscribers are higher than the respective curve for non-subscribers. The travel-time difference decreases as the market saturation increases.

Figure 2.21 shows this result more clearly. During the peak of the rush hour, a ten percent market saturation yields an average improvement of 200 seconds which corresponds to 28%! Of course, this statistic does not prove the overall benefit of such a service because the improvement could be strongly biased towards a small subset of subscribers enjoying extreme reductions in travel-time while others have no benefit or are delayed. Still, Figure 2.21 may be important for marketing purposes since it clearly shows that the system works well if restricted to a small clientele.

It is interesting to take a look at the distribution of relative delays upon arrival at the destination. From the five runs that were executed for market saturation $m = 0.2$ we gathered the subscribers into three groups (see Figure 2.22): those who were never re-routed ($rr = 0$), those who were re-routed one to five times ($rr = 1$), and those who were re-routed six to ten times ($rr = 2$). Trips with more than ten re-routing events were not considered because of their insufficiently small counts. The curve for $rr = 0$ is mainly a normal distribution centered around zero delay since this subset does not differ from non-subscribers at all except for the fact that they "know" they cannot receive better alternative routes. The curve for $rr = 1$ that also peaks at zero delay, is biased towards

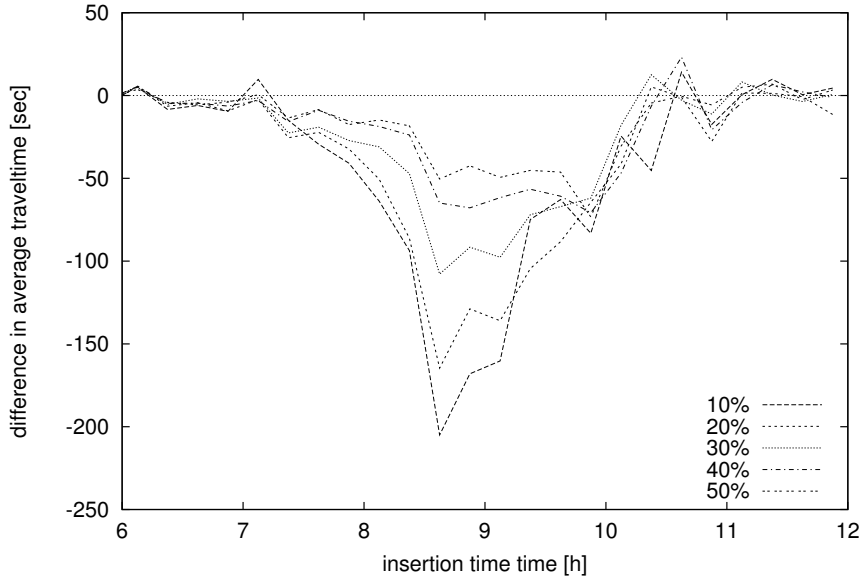

Figure 2.21: *Difference of average travel-time between subscribers and non-subscribers (iteration 20)* — Negative values denote shorter travel-times for subscribers. The advantage of subscribers over non-subscribers decreases as the market saturation grows.

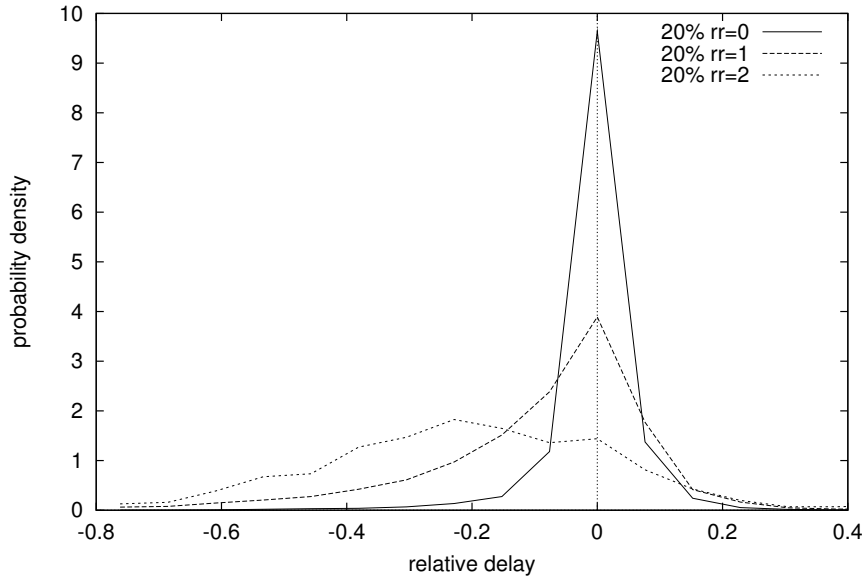

Figure 2.22: *Quality of service for $m_{o-l} = 0.2$ (iteration 20)* — The curves show the relative delay of re-routed vehicles for different re-routing counts. The more often a vehicle is re-routed the more the distribution is biased towards negative delays: it pays to be re-routed several times.

negative delays. Specifically, there are more subscribers arriving earlier than non-subscribers. The effect becomes even more obvious for curve $rr = 2$, where the peak has actually shifted towards negative delays. This means that within our re-routing approach *it is worth while to be re-routed more than once.*

Chapter 3

Implementation

3.1 General Overview

The implementation of PAMINA III is based upon descendent C++ classes derived from the C++ base classes provided in the Parallel Toolbox Version 2.0. A detailed description can be found in [39]. However, we would like to outline how the traffic simulation interacts with the toolbox and its basic functional elements (refer to Figure 3.1). The PAMINA source code splits into three major parts: (a) the underlying CA model, which is the most compact module containing only

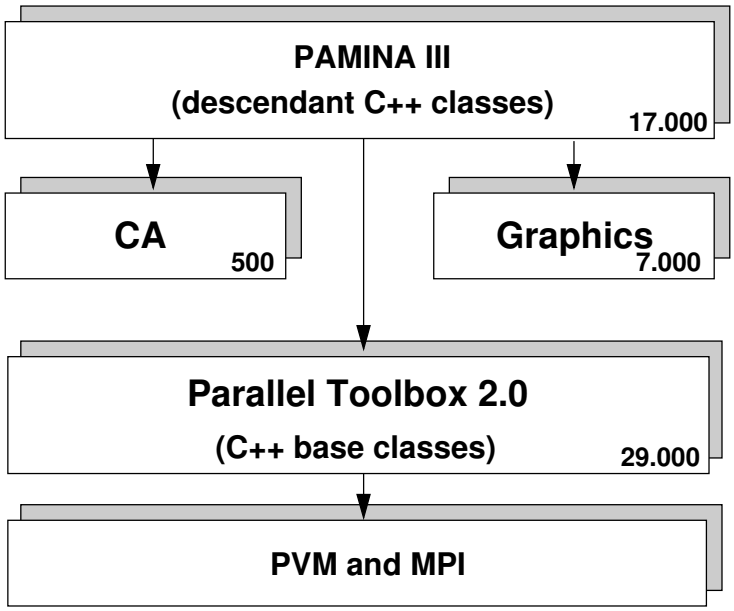

Figure 3.1: *Software implementation structure of PAMINA III* — The parallel toolbox serves as an interface between the traffic application classes in PAMINA and the message passing libraries PVM and MPI. The figures in the lower right corners denote the number of source-code lines in each module. Note that most of the programming effort was required for coding the network implementation.

about 500 lines of code, (b) the graphics support module with about 7.000 lines, (c) handling of the traffic network elements, the route-plans, and statistics with about 17.000 lines of code. The latter also contains the interface to the Parallel Toolbox which by itself has about 29.000 lines of code. The toolbox uses the commonly available message library PVM 3.3.1 as high-level communication interface to the underlying computer hardware. PAMINA II and the Parallel Toolbox 1.0 have been ported to several platforms, such as Sun Solaris, SGI Irix, DEC Alpha, IBM RS6000, and PC Linux.

3.2 Parallelization

The inherent structure of a traffic micro-simulation favors a *domain decomposition* as the general approach to parallelization:

- The street network can easily partitioned into tiles of equal or almost equal size. A realistic measure for size is not the number of net elements (nodes and segments), but the CA grid lengths associated with those elements (see Figure 3.2 for a schematic view of tiles and Figure 2.3 for a concrete example). Tiles are then assigned to processors.
- The range of interdependencies between network elements are restricted to the interaction range of the CA. All current rule sets have an interaction range of either $v_{max} \equiv 35[m]$ or $2v_{max} \equiv 70[m]$ which is a short distance compared to the average length of the edge segments (e.g. $484[site] \equiv 3630[m]$ for map FRG) in a motorway network. In a city street network which has somewhat shorter segments on the average, there may be links which are to short to hold at least 10 grid sites. We artificially enlarge the links to contain the minimum number of sites. In any case, the most straightforward approach is to cut the network at the middle of street segments.
- As a consequence of the distribution the tiles exchange *boundary information* containing all vehicle data necessary for the execution of the traffic rule set, resulting in local communication between neighboring tiles. Obviously, there is no real counterpart for a boundary in the original traffic simulation. It is an artifact of the parallel implementation. We differentiate between two resolutions of vehicle data: (a) the *primary vehicle data* only contains information on the location of vehicle, which is required for the CA rule set. (b) The *secondary data* contains all other information about the vehicle, such as maximum velocity and route-plan.

The traffic links and intersections are mapped onto the structural elements supplied by the toolbox. These are *nodes*, *edges*, and *boundaries*:

- The *node* class was used to represent exactly one node of the traffic network. The toolbox guarantees that nodes exactly reside on one CPN. This is advantageous for the substructures associated with a node: all elements can assume that other related elements of the same substructure reside on the same CPN. If an incident edge happens to be split (see below), at least the half of the edge next the node can be assumed to be local.

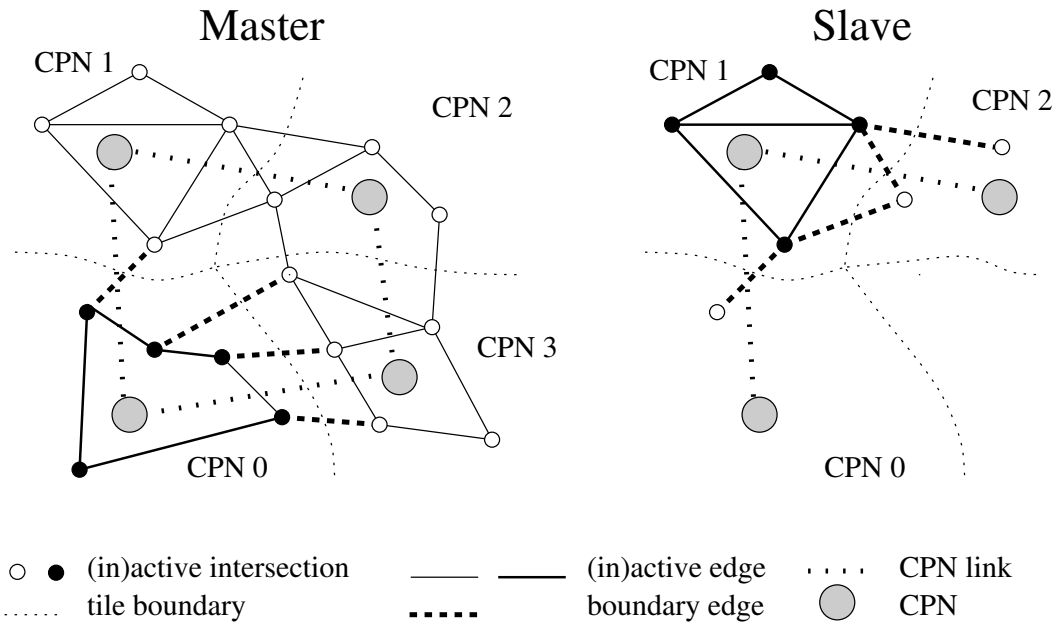

Figure 3.2: *Geometric distribution of a street-network* — LEFT: Traffic network on the master: inactive representation of the complete network. Active representation of the local sub-net. RIGHT: Traffic network on a slave (here CPN 1): only active representation of the local sub-net.

- The *edge* was used to represent a bi-directional multi-lane street segment. For each direction a multi-lane CA grid was used. In contrast to nodes, an edge may be duplicated by the toolbox in case that the incident nodes reside on different CPN. Such a so-called *boundary-edge* or *inter-CPN edge* is split exactly in the middle. A discrete CA grid of odd length l has to be handled with care by assigning $\lfloor l/2 \rfloor$ sites to one and $\lceil l/2 \rceil$ sites to the other CPN after breaking their symmetry. On one CPN the first half is active and on the other CPN the second half.

- Exactly in the middle *boundaries* are retrieved from the grids and transferred to the remote CPN. They contain information about the state of the traffic system close the split point.

3.2.1 Initial domain decomposition

The toolbox handles the initial distribution and subsequent load balancing if requested. The geometric node locations are used to perform a recursive orthogonal bisection of the traffic network.

For the initial distribution of a network with nodes $n_1 \dots n_N$ onto CPNs $C_1 \dots C_C$ we have to make three assumptions:

- We have performance values for each CPN given in arbitrary but proportional values $S_1 \dots S_C$ where larger values denote better performance.

- Each node n_i has an estimated load l_i associated with it which is derived from the complexity of the node itself and all its incident edges. In particular, we use the number of CA grid–sites on transfer lanes at junctions as a measure for the nodes, and the number of CA grid-sites on traffic links as a measure for the edges. In PAMINA III the actual execution time of the previous iteration is used individually for each network element (also see 3.2.5).

- Each node n_i has an Euclidean location (x_i, y_i).

Recursive algorithm

The algorithm is defined recursively for a set of nodes $n_p \dots n_q$, a set of CPNs $C_a \dots C_b$, and recursive depth d.

1. If the number of CPNs $b - a + 1$ is one, assign all nodes to this CPN.

2. Otherwise, split CPNs in halves $C^l = \{C_a, \dots, C_{\lfloor a+b/2 \rfloor}\}$ and $C^r = \{C_{\lceil a+b/2 \rceil}, \dots, C_b\}$ with sum performances $S^l = \sum_{i=a}^{\lfloor (a+b)/2 \rfloor} S_i$ and $S^r = \sum_{i=\lceil a+b/2 \rceil}^{b} S_i$.

3. Sort nodes according to their x-coordinates for even depth d and according to their y-coordinates for odd depth d.

4. Split nodes at node n_j $(p < j \leq q)$ into two sets $N^l = \{n_p, \dots, n_{j-1}\}$ with load $L^l = \sum_{i=p}^{j-1} l_i$ and $N^r = \{n_j, \dots, n_q\}$ with load $\sum_{i=j}^{q} l_i$ in such a way that their load ratio is equivalent to the ratio of the performance values:

$$\frac{L^l}{L^r} \stackrel{!}{\simeq} \frac{S^l}{S^r}$$

   Due to the granularity of the l_i exact equality may not be achieved. In this case, j has to be chosen in such a way that the inequality in minimized.

5. Split the subsets C^l with nodes N^l and C^r with nodes N^r recursively.

Corrections

Since no topological aspects are considered, the resulting tiles may not be connected anymore. Nevertheless, each CPN can re-establish a single connected component by casting off all superfluous, not connected components to neighbors and keeping the largest one only.

If the area of the map is not square, the first split of the nodes should be parallel to the *shorter* edge of the rectangle. This way, unnecessarily narrow stripes are avoided.

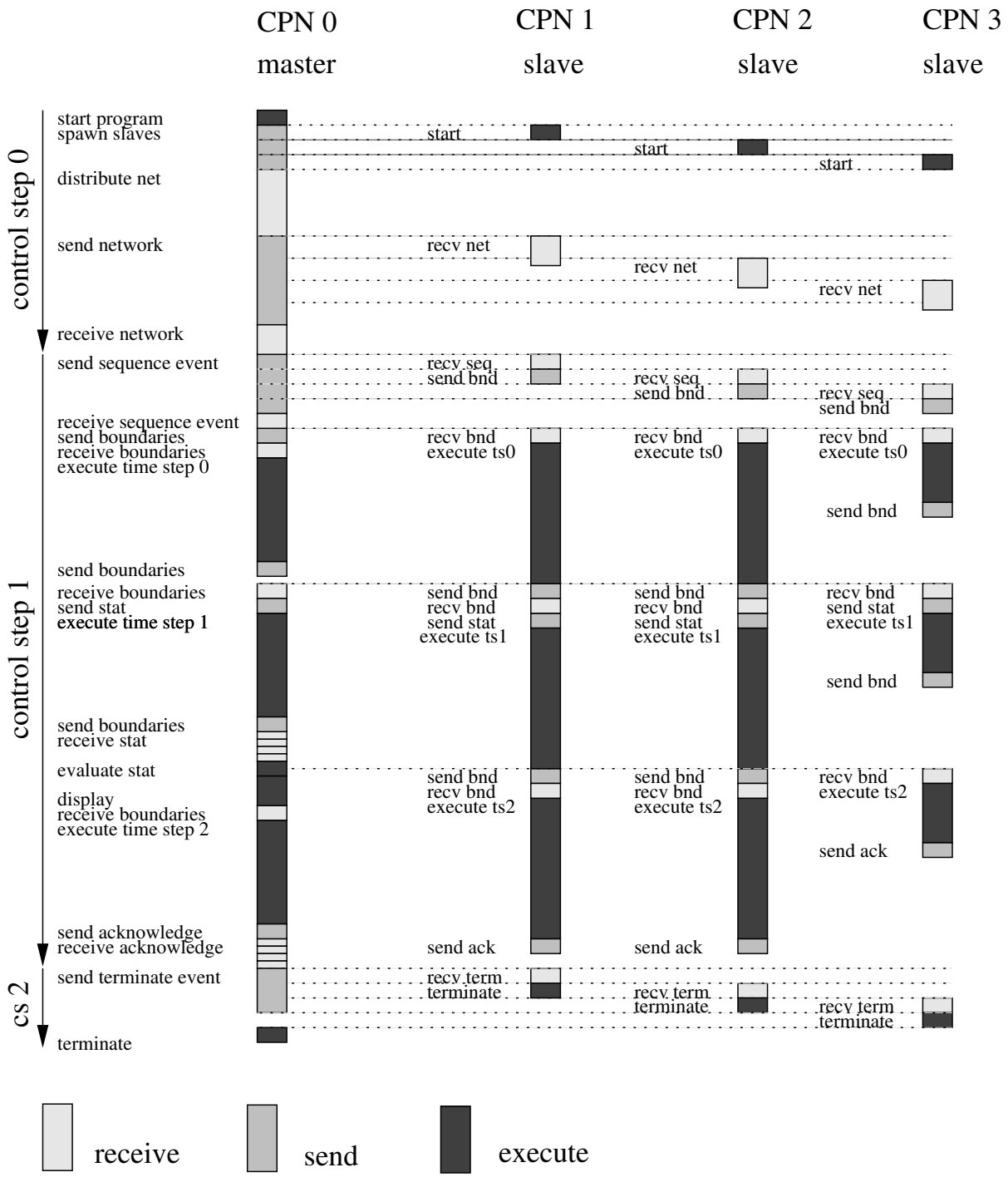

Figure 3.3: *Timing of a simulation run*

3.2.2 Simulation control

The toolbox uses a master-slave algorithm as control logic for program execution. The master process is started first, usually directly by the user. It spawns several slaves on the parallel computer architecture. For the simulation itself the master also operates as a slave. It does, however, also retain some additional functionality.

The following enumeration is supposed to convey an idea of the main steps executed during a simulation run. See Figure 3.3 schematic overview of such a run on four CPNs.

1. The user initializes PVM on the future master CPN by calling the interactive PVM-shell `pvm`. He adds all CPNs manually or starts a script to do this automatically. The user starts the application executable which is the master process.

   PAMINA III also supports MPI as message passing library, in which case the user simply calls the application executable with one additional command line parameter (`-np`) defining the number of CPNs to be used.

2. The master starts all other instances of the program on the slave CPNs. They will enter the main message loop and wait for messages.

3. The master reads the network structure from the input data files.

4. The master distributes the network and sends out messages to the slave CPNs with encoded network elements.

5. The master sends an event to all CPNs to start a simulation sequence of a given number of time-steps. All slaves send out the boundaries for the first time-step to the neighboring CPNs. During that sequence the master mainly functions as a slave.

6. The slaves enter the main loop:

   - They wait for the arrival of all boundaries from their neighbors.
   - If necessary, statistical data (e.g. idle time statistics) is sent out.
   - If idle-time statistics are available local load balancing is done.
   - They execute a time-step.
   - Unless the end of the sequence is reached they send out the boundaries for the next time-step.

   The master has several additional functions:

   - If available, global statistical data from the slaves is processed and displayed.
   - The X-Windows event queue is checked if there is need to update the graphics output.
   - The PVM environment is checked for changes of the CPN topology. If necessary, CPNs may be removed from the topology or new CPNs may be added to the topology.

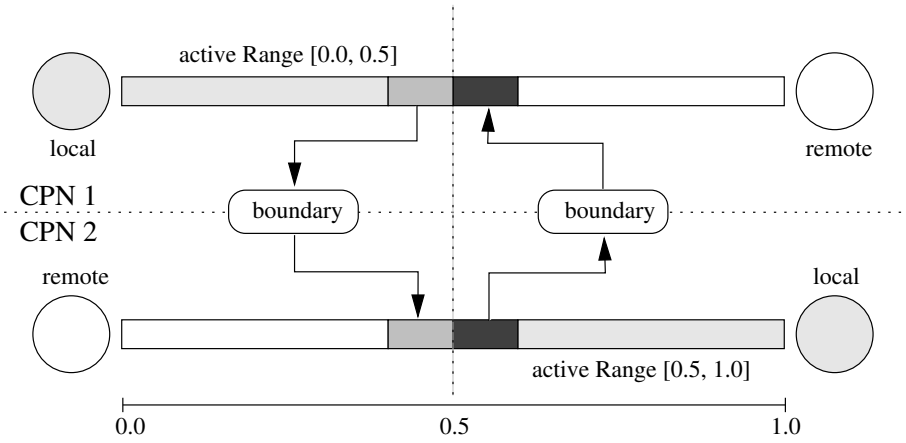

Figure 3.4: *Exchanging boundaries*

7. The master stops the simulation by sending an termination event to all slaves.

8. The master and the slaves stop execution.

3.2.3 Boundaries

After the distribution of the nodes of the network there will be edges crossing CPN boundaries which are called *boundary edges*. They are cut exactly in the middle so that each associated CPN computes half of the edge. Note that, therefore, boundary edges exist twice (see Figure 3.4). Before either CPN can execute a time-step it obtain information about the objects on the remote CPN. We differentiate between *primary* and *secondary* vehicle data. Primary data only contains information about the *location* of the any objects in the boundary area, which is sufficient for evaluating the CA rule set. Secondary data also contains information about (or rather copies of) the vehicles themselves.

As for the width of the boundary we have to transmit information about all vehicles within the *interaction range*. This requires encoding and decoding of all vehicle data that are located within a range of v_{max} sites from a boundary. The information is transferred to the remote CPN. Over there, it is given to the duplicate of the edge which appends this information to the local data stored on the edge. The additional data allows the execution of the time-step.

The actual size (byte-wise) of boundaries can be optimized by taking advantage of specific characteristics of the CA rule set. If the local density is high, that is, the boundary is located within a traffic jam, there may be more than one vehicle per lane. The CA rules, however, only refer to the *immediate* predecessor or successor on each lane, reducing the maximum number of vehicles in a boundary to one per lane. Moreover, only the *primary* vehicle data is needed and not the secondary data including the route-plan. This is true, at least for the first sub-time-step. In the second sub-time-step all vehicle data is needed to guarantee a consistent vehicle migration across CPN boundaries, which will be described next.

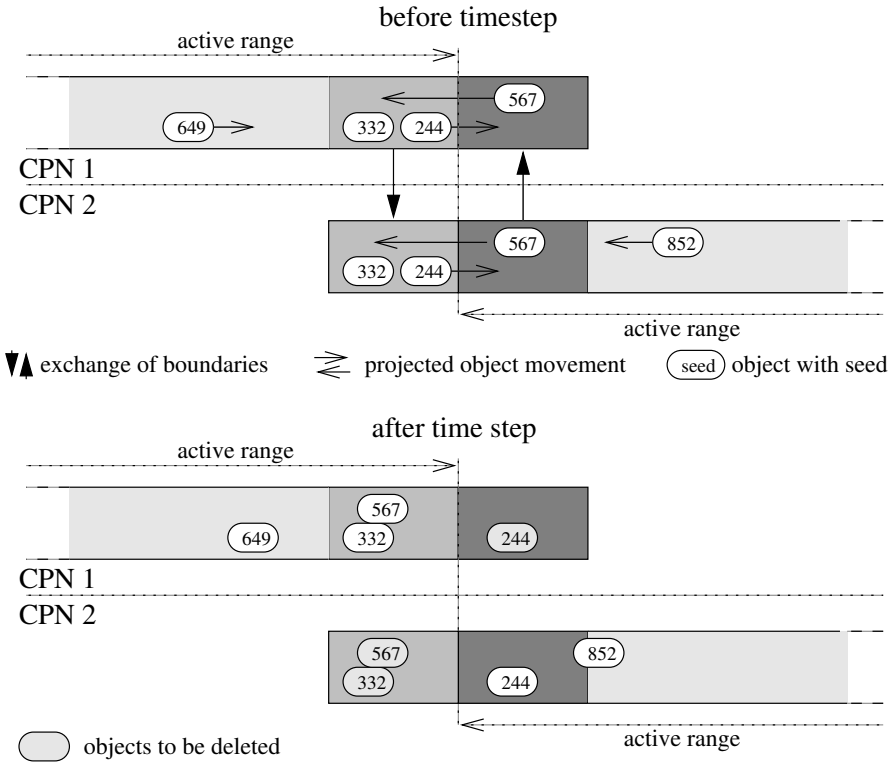

Figure 3.5: *Consistent handling of boundary objects*

Consistent handling of boundary objects

Boundaries contain regions in which the same objects are handled by both CPNs (see Figure 3.5). In order to guarantee consistence of the simulation it is necessary that both instances of the same object behave exactly *identically*. In case of a deterministic simulation this is, of course, no problem. In case that decisions of objects depend on random numbers it is necessary to make the random generator[1] part of the object and to pass it together with the object data to the remote CPN. In the CA model the stochasticity of the motion of a vehicle is completely determined by a single bit which is set with a probability of p_{dec}. In PAMINA II this bit is included whenever secondary boundary information is transferred.

During a time-step the objects on an edge usually change positions, that is, some will probably leave the remote boundary and enter the normal active part of the edge whereas others will do vice versa (see Figure 3.5). After the time-step all objects have to be deleted that still remain in boundaries supplied by remote CPNs (object 244 for CPN 1 and objects 332 and 567 for CPN 2). Likewise all objects that have entered the local active part of the edge have to be permanently inserted (object 567 for CPN 1 and object 244 for CPN 2). Note that object 332 has to be deleted, too, although it would be at the correct location for the next time-step. Leaving it in the edge, however, would

[1] or rather: values necessary to reproduce the random sequence on the remote CPN, which could be for example the current seed of the generator

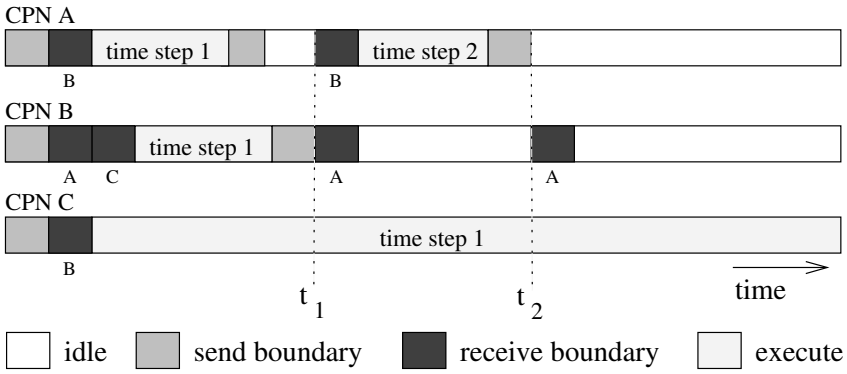

Figure 3.6: *Timing of boundaries*

lead to collision with the boundary for the next time-step which will contain another copy of that object.

3.2.4 Timing of a simulation time-step

The simulation uses a parallel update with a global time-step. However, synchronization of all CPN is only performed after a so-called *simulation–sequence* comprising approximately 10-20 time–steps. In between, there is only a weak synchronization through the exchange of boundaries. Between neighboring CPNs there may be a difference in time-steps of $\Delta t = \pm 1$ as displayed in Figure 3.6: due to slow execution of time step 1 on CPN C, CPN B has received boundaries from CPN A for time step 2 *and* already for time-step 3. The toolbox buffers those early boundaries automatically.

The global time–step is used to guarantee consistent collection of statistical data: Although partial results from the CPN may not be collected at the same physical wall-clock time due to a potential time-step gradient (see [28]), they always belong to the same logical time-step. The master CPN takes care of combining partial results.

Each global time-step is subdivided into two sub-time-steps. The first sub-time-step is used for lane changing, while the second sub-time-step is used for forward motion. Each sub-time-step requires the exchange of boundaries between CPN, although they are of different resolution: the first time-step only requires the transfer of primary vehicle data, while the second sub-time-step also comprises the secondary data.

Each sub-time-step is subdivided into a preparation phase (P) and an execution phase (E) preceded by the implicit local synchronization (IS) through boundary exchange as summarized in Table 3.1.

3.2.5 Benchmarks

We used a route-plan of run 11 (iteration 60) to measure the real-time-ratio of PAMINA. The simulation ran on SUN Enterprise 2000 with 14 CPUs (250 MHz) and 2 Gigabytes of memory. Figure 3.7 shows the results for different numbers of CPUs (4, 6, 8). The maximally obtainable

sub-time-step	IS/P/E	Action
1	IS	exchange primary vehicle data, gather statistics
1	P	CONN, DR, MR, resolve dead-locks
1	E	lane change
2	IS	exchange all vehicle data
2	P	CONN, ER, AR
2	E	motion, migration

Table 3.1: *Timing of a simulation time-step*

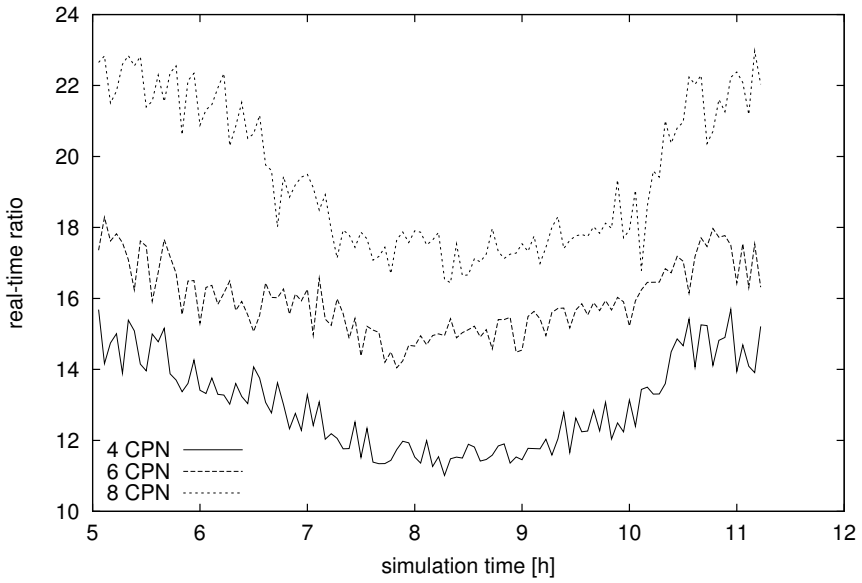

Figure 3.7: *Performance of PAMINA III* — The real-time ratio is plotted for 4, 6, and 8 CPUs on a Sparc Enterprise 2000 with 250 Mhz. The trough around 8:00 am is caused by the high vehicle density inside the study-area.

ratio for PAMINA is about 22 for 8 CPUs and early simulation hours while the study-area is still empty. The RTR drops to 18 during rush-hour at 8:30 am.

In PAMINA III we implemented simple external feedback for the initial static load balancing. During run time we collect the execution time of each link and each intersection. The statistics are dumped to file every 1000 time-steps. For the next iteration run the file is fed back to the initial load balancing algorithm. In this iteration, instead of using the link lengths as load estimate, the actual execution times are used as distribution criterion.

To verify the impact of this approach we monitored the execution times per time-step throughout the simulation period. Figure 3.8 depicts the results of run 17 for several iterations. For iteration 1, the load balancer used the link lengths as criterion. The excution times were low until the first grid-locks appear around 7:30 am. The execution time increased fivefold from 0.04 [sec] to 0.2 [sec]. In iteration 2 the execution time is almost independent of the simulation time. Note that due to the

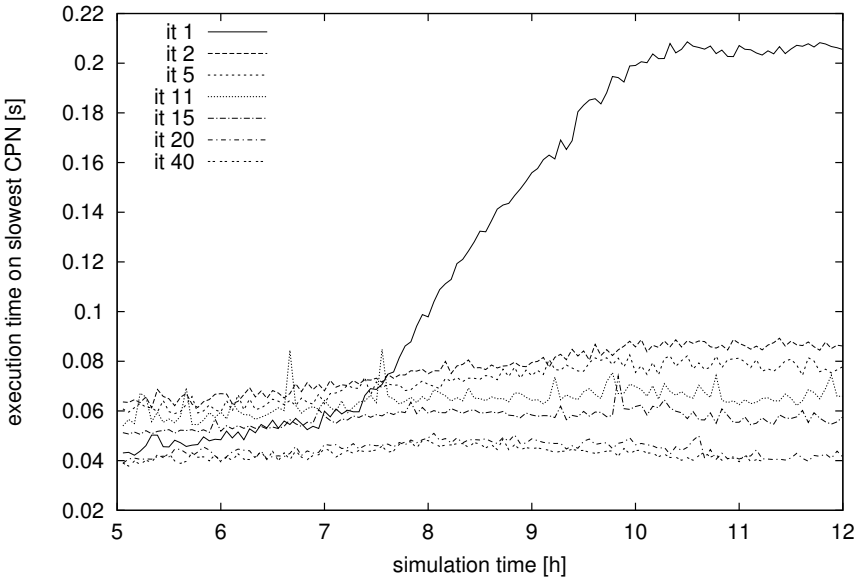

Figure 3.8: *Execution times with external load feedback* — The feedback visibly improves the performance of the simulation. After only two iterations the execution time is almost time-independent.

equilibration the execution time for early simulation hours increased 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, 20, and 40). The improvement of execution times is mainly due to the route adaptation process: all grid-locks have disappeared and the average vehicle density is much lower.

Chapter 4

Installation & Compilation

4.1 Installing the Files

4.1.1 Installing the Simulation Suite

All files required by the core simulation suite are contained in the GNU-zipped `PaminaIII-<VERSION>.tgz` archive. Unpack this archive into a directory of your choice. The subdirectory `PaminaIII` will be called the *PAMINA home directory* from here on.

4.1.2 Requirements

The source code of the microsimulation depends on a handful of other tools and libraries. Most of them are available as open source packages on any unix system. These are:

- GNU g++
- GNU zip
- GNU make
- GNU m4
- GNU bison
- GNU flex
- Python

For generating the statistical diagrams from the simulation output one optionally needs the following tools:

- GNU awk
- Gnuplot
- ppmtogif

For the tools listed above any up-to-date version currently available should work fine. You can always check the version numbers used for PAMINA development at the PAMINA homepage:

`http://www.the-rickerts.de/mr/PAMINA.html#prerequisites`

However, the core library for the parallization may be harder to come by. This is either

- Parallel Virtual Machine (PVM) `http://www.epm.ornl.gov/pvm/pvm_home.html` or
- Message Passing Interface (MPI) `http://www-unix.mcs.anl.gov/mpi/` .

The home page of PVM has been down for a while. That is why a tar archive of version 3.4.4 has been included in the `3rdparty` subdirectory of the Pamina archive.

The microsimulation suite has been tested for a few platforms and architectures. The most up-to-date information on compatible platforms can be found at the PAMINA homepage

`http://www.the-rickerts.de/mr/PAMINA.html#platforms`

4.2 Compilation

4.2.1 Environment Variables

Before you compile the suite make sure to set the following environment variables. If possible include them into your shell profile script.

`SITE` Choose a name for the computer site that you build the simulation on. It will be used to include site specific settings in the `config` subdirectory. See 4.2.2.

`COMMUNICATION` Choose the core library for the parallization of the microsimulation. This has to be either `PVM` or `MPI`. Note that due to the importance of the setting most of the scripts will not start without this variable having been correctly set.

`HOST` This variable should contain the hostname of the computer you compile on. It is usually set automatically in your profile.

`PAMINA_ARCHIVE_HOME` Set this variable to a directory that will contain the gnu-zipped files of the microsimulation. We recommend to make it point to a large but not necessarily fast file system.

`TOOLBOX_APP_HOME` This setting is optional. TODO.

There are more settings related to the parallel core library you you choose. Please, see 4.2.4 for PVM or 4.2.5 for MPI.

4.2.2 Site Specific Settings

Your computer site may require specific settings (e.g. choosing a specific version of the GNU compiler or setting a specific path to a system library). All this customization should be made in a file called `<SITE>.Settings` which must be located in the `config` subdirectory of the PAMINA home directory. If available the file will automatically sourced by the make process.

4.2.3 Hardcoded Simulation Parameters

The microsimulation has several hard coded limits which can be set by C-defines in the source code. The most interesting are:

`MAX_LANES in TrafficData/CAConfig.h` Sets the maximum number of lanes that can be used in the microsimulation segments.

`CA_DEFAULT_V_MAX in TrafficData/CAConfig.h` Sets the default maximum speed of the vehicles in CA units.

`MAX_ROUTE_LENGTH in TrafficData/TRouteplan.h` Sets maximum number of links in a route plan.

`MAX_SITES_PER_GRID in TrafficData/CAConfig.h` Sets the maximum number of grid sites per lane in a street segments and hence the maximum legth of a street segment.

`SP_NAME_SOURCETIMESLACK in TrafficData/TrafficDataConfig.h` Sets the time slack in seconds with which a late insertion of a vehicle is still regarded to be *in time*.

Note that this list is far from complete. TODO!

4.2.4 Setting Up PVM For Compilation

To use PVM as the core parallelization library follow these steps:

1. Retrieve a current version of the library from the web site

http://www.epm.ornl.gov/pvm/pvm_home.html

or use the archive provided in the `3rdparty` subdirectory.

2. Unpack the archive in a directory `DIR` of your choice.

3. Set the environment variable `PVM_ROOT` to the root directory of PVM. For example, on a bash/ksh shell use

```
PVM_ROOT=DIR/pvm3
export PVM_ROOT
```

We recommend to include the settings into your profile. Make sure that you also browse through the Readme file located in the `pvm3` subdirectory.

4. `cd` into `$PVM_ROOT`

5. Type `make`

6. Set the environment variable `PVM_ARCH` using a script provided by PVM. For example, on a bash/ksh shell use

```
PVM_ARCH=‘$PVM_ROOT/lib/pvmgetarch‘
export PVM_ARCH
```

We recommend to include the settings into your profile.

7. Set the environment variables for the execution path and the manual pages:

```
PATH=$PATH:$PVM_ROOT/lib/$PVM_ARCH
PATH=$PATH:$PVM_ROOT/bin/$PVM_ARCH
PATH=$PATH:$HOME/pvm3/bin/$PVM_ARCH
export PATH
MANPATH=$MANPATH:$PVM_ROOT/man
export MANPATH
```

We recommend to include the settings into your profile.

4.2.5 Setting Up MPI For Compilation

To use MPI as the core parallelization library follow these steps:

1. Retrieve a current version of the library from the web site

http://www-unix.mcs.anl.gov/mpi/

2. Unpack the archive and follow the instructions provided in the archive. MPI is a well documented library so we won't provide any additional information here.

3. Set the environment variable `MPI_ROOT` to the root directory of MPI. This is just one level above the selected MPI device. For example, if your MPI device installation is in `/vol/mpich/ch_p4` on a bash/ksh shell use

   ```
   MPI_ROOT=/vol/mpich
   export MPI_ROOT
   ```

   We recommend to include the settings into your profile.

4. Set the environment variable `MPI_ARCH` to reflect your system architecture. You can choose any name. It will be used to differentiate between versions of PAMINA compiled for different archtectures. On a linux system a sensible choice would be `LINUX`. Also set the environment variable `MPI_DEVICE` to the MPI you have compiled MPI for. For example, on a bash/ksh shell use

   ```
   MPI_ARCH=LINUX
   export MPI_ARCH
   MPI_DEVICE=ch_p4
   export MPI_DEVICE
   ```

   We recommend to include the settings into your profile.

5. Set the environment variables for the execution path and the manual pages:

   ```
   PATH=$PATH:$MPI_ROOT/$MPI_DEVICE/bin
   export PATH
   MANPATH=$MANPATH:$MPI_ROOT/$MPI_DEVICE/man
   export MANPATH
   ```

   We recommend to include the settings into your profile.

Note that you can setup MPI for more than one device. For PAMINA, however, exactly one device only must be selected using `MPI_DEVICE`.

4.2.6 Starting the Compilation

At this point we assume that you have installed all required tools and made all required changes to your environment as described above. To make build all applications of the suite follow these steps:

1. `cd` into the PAMINA home directory.

2. Type

   ```
   make prepare
   make depend
   make
   ```

   Note that on your system the name of the GNU make may be different (e.g. `gmake`).

If you run into any errors executing the steps above you may have to edit your site specific settigs file in the `config` subdirectory. See 4.2.2. The command `make prepare` will have created a sample file for you by that time.

Chapter 5

Usage

In this chapter we will explain the usage of the microsimulation suite PAMINA. Note that this mainly covers the execution of pre-defined scenarios and the control script `RunIterativeReplanning.py`. Although it is possible to start the planner, the route converter or the microsimulation directly this is not recommended.

5.1 Scenarios

A scenario is tested setup for PAMINA which allows you to check the correct installation of the suite and at the same time serves as an example on how to configure the suite according to your needs.

Each scenario usually comprises the following components:

- network files,
- an initial routefile,
- a makefile,
- a Pamina configuration,
- a set of Gnuplot evaluation files.

Note that if you run into trouble executing one of the scenarios it is wise to double check your setup and consult the author of Pamina before trying to build your own scenario from scratch.

5.1.1 Installing a Scenario

A scenario also comes in a GNU-zipped archive. The archive should be unpacked in the `scenarios` subdirectory of your PAMINA home directory. Note that some of the scenarios may consume a lot of space on your hard drive. You may want to make `scenarios` a link to large file system.

5.1.2 Setting Up PVM For Execution

If you use PVM as the core parallelization library follow these steps:

1. Make sure that your environment contains all variables for PVM that were described in 4.2.4.
2. If you haven't do so already, create a *hostfile* in your home directory which reflects your host setup. Make sure to include
   - one regular line of format `& HOSTNAME` for each host to be included into your parallel machine, and
   - one special comment of format `#! HOSTNAME NUMBER-OF-CPUs` for each host having more than one CPU.

   There is a sample hostfile `hostfile.sample` in directory `etc` which can serve as a template. Note that you can have more than one hostfile. However, if you do not use `$HOME/hostfile` you have to change the setting in the PAMINA configuration file (see TODO).
3. Start the PVM daemon process by typing

   ```
   pvm ${HOME}/hostfile
   conf
   quit
   ```

   Make sure the output of the `conf` command reflects the settings in your hostfile. You should see someting like this:

   ```
   mr@casimir:~$ pvm ${HOME}/hostfile
   pvm> conf
   conf
   1 host, 1 data format
      HOST     DTID     ARCH   SPEED       DSIG
   casimir    40000    LINUX     350 0x00408841
   pvm> quit
   quit

   Console: exit handler called
   pvmd still running.
   ```

5.1.3 Setting Up MPI For Execution

Currently, MPI is only supported for multiple CPNs on the same host using `mpi_run`. That is why there is no need to do additional configuration beyond the settings of environment variables as described in section 4.2.5.

5.1.4 Running the Scenario

To run a scenario follow these steps:

1. Make sure you have set up all environment variables described in 4.2.1.
2. Make sure you have set up the core parallelization library for execution. See 5.1.2 for PVM or 5.1.3 for MPI.
3. `cd` into the directory of scenario the name of which starts with `PaminaScenario...` and which is a subdirectory of the `scenarios` subdirectory of the Pamina home directory.
4. Type `make prepare` to create all neccessary directories and files. This will usually create the subdirectory `log` and a link called `archive` into your `PAMINA_ARCHIVE_HOME`. Note that the name of the GNU make utility may be named differently on your system (e.g. `gmake`).
5. Type `make run` to start the iterative replanning and microsimulation. If you run into any problems at this point please check your setup and if everything fails consult the author.
6. During the execution of the suite you can monitor the progress by looking at the files in the `data` and `archive` directories. The standard output and standard error of the applications can be found in the `log` subdirectory.

If the suite returns successfully and you have gnuplot and awk installed on your system you can have look at the graphical output of the iterative replanning process. To do this type `make routing_stat`. Usually there is more than one gnuplot file available. To view any of the statistics in the `evaluate` subdirectory simple type `make GNUPLOT-FILENAME-WITHOUT-EXTENSION` for any file that you see in the subdirectory.

5.2 Input Files

In this section we describe the structure of all input files. They belong to the following groups:

- Files containing the configuration of the parallel computing environment and the applications.
- Files containing the street nework elements
- The file containing the route set which is to be executed.

5.2.1 Configuration

Parallel Machine Setup

The setup of the parallel machine depends on your choice of core parallelization library. Please, see 5.1.2 for PVM and 5.1.3 for MPI.

Application Settings

All configurable application settings are contained in a ASCII configuration file which resembles a Windows INI file. The control script reads a configuration file template, adds items to it and writes out the final version which is read by the route converter and the microsimulation.

Currently, this manual does not cover all settings that can be made in the configuration file. For a complete list see the commented file `ParadoxOriginalIterationConfig` of the Braess paradox scenario.

The settings which most likely need modification follow:

`PVM.Hostfilename` Set the name of the PVM hostfile which will be used to define the parallel machine. The filename may contain variables as described in 5.2.2.

`Iteration.PaminaHomeDir` Set the directory where the PAMINA simualation suite was installed, e.g. the one which has the `src` and `bin` subdirectories. It's a good idea to use a relative path. The directory path may contain variables as described in 5.2.2.

`Iteration.ArchiveDir` Set the archive directory where files will be archived. The filesystems pointed at has to be large enough to accomodate all file of the iteration process. For example: Each iteration of the Dallas Fortworth study area scenario generates about 120 Megabytes of data (mostly gnu-zipped).

`TrafficData.BaseFilename` Set the base filename (including the path but excluding the extension) of the network files in the PAMINA format. The filename may contain variables as described in 5.2.2.

`TrafficData.LoadAccessories` If set to 1 this switch will activate the loading of the network accessories.

`TrafficData.LoadPhases` If set to 1 this switch will activate the loading of the intersection phasing data.

`TrafficData.StartTime` Sets the start time of the simulation in hours.

`Planner.TransimsMapBaseFilename` Set the base filename (including the path but excluding the extension) of the network files in the TRANSIMS format. The filename may contain variables as described in 5.2.2.

`Planner.ReplanningFraction` Sets the fraction $[0 \ldots 1]$ of routes which need to be replanned. Note that during the first iteration (with index 1) the replanning fraction is set to 1.0 which will trigger the replanning of the whole route file.

`RouteConverter.Filename` Set the template file name that will be used for the TRANSIMS route file during the iteration. A good choice would be

`%W/data/plans.TRANSIMS.`*SCENARIO*`.__ITERATION__`

The `%W` will automatically be replaced by the work directory and the `__ITERATION__` by the index of the previous/current iteration. Also see 5.2.2.

`RouteConverter.MinTime` Sets the minimum departure time of a route in hours. Any route before that time will not be transferred to the PAMINA route file.

`RouteConverter.MaxTime` Sets the maximum departure time of a route in hours. Any route after that time will not be transferred to the PAMINA route file.

`Router.Routeplan` Set the template file name that will be used for the clipped PAMINA route file during the iteration. A good choice would be

`%W/data/plans.PAMINA.`*SCENARIO*`.__ITERATION__`

The `%W` will automatically be replaced by the work directory and the `__ITERATION__` by the index of the previous/current iteration. Also see 5.2.2.

`ParSim.MaxTimeSteps` The maximum number of timesteps (seconds) that the microsimulation should execute. Make sure to add one timestep to get the statistical data of the final timestep. Note that the simulation will also terminate after a certain fraction of the routes have been executed.

`ParSim.Graphics` If non-zero the microsimulation will try to display an X window. The graphical output only works if PAMINA has been compiled using setting `XWINDOWS=1`. See section 5.5.

5.2.2 Vaiable Substitution in Filenames

Most of the filenames in the configuration file may contain specific variables which are replaced by default paths at runtime. These are:

`%h` Home directory of the user running the application.

`%W` Working directory of the application.

`__ITERATION__` Current iteration usually formatted as a three digit decimal with leading zeros.

5.2.3 TRANSIMS Network Elements

The TRANSIMS network files define the street network that is used by the planner `planner`. All files described here use a line oriented format with one data record per line. The first line contains the column names and is to be ignored. Otherwise, there are no comments lines. Columns are seperated by commas.

Nodes (.nod)

The format is described in Table 5.1. The base filename (without the extension) is given by configuration setting `Planner.TransimsMapBaseFilename` in 5.2.1.

column	format	description
1	LONG	Logical node id.
2	DOUBLE	x coordinate given in Meters.
3	DOUBLE	y coordinate given in Meters.

Table 5.1: *Input file format for TRANSIMS network nodes* — See table for details on columns.

Links (.edg)

The format is described in Table 5.2. The base filename (without the extension) is given by configuration setting `Planner.TransimsMapBaseFilename` in 5.2.1. TRANSIMS links are bidirectional by default. To activate a direction the corresponding number of lanes must set to zero.

5.2.4 PAMINA Network Elements

The PAMINA network files define the street network that is used by the microsimulation PAMINA. All files described here use a line oriented format with one data record per line. Columns are seperated by white characters.

Nodes (.nod)

This file contains the nodes (intersections) of the street network. The format is described in Table 5.3.

Links (.edg)

This file contains the links (street segments or edges) of the street network. The format is described in Table 5.4.

column	format	description
1	LONG	Logical link id.
2	LONG	Logical id of node A
3	LONG	Logical id of node B
4	LONG	Number of lanes from B to A
5	LONG	Number of lanes from A to B
6	DOUBLE	Length of the link in Meters.
7	DOUBLE	Capacity of the link from B to A in vehicles/second
8	DOUBLE	Capacity of the link from A to B in vehicles/second
9	DOUBLE	Speed limit of the link from B to A in meters/second
10	DOUBLE	Speed limit of the link from A to B in meters/second
11	DOUBLE	Free speed of the link from B to A in meters/second
12	DOUBLE	Free speed of the link from A to B in meters/second
13	DOUBLE	Crawl speed of the link from B to A in meters/second
14	DOUBLE	Crawl speed of the link from A to B in meters/second
15	LONG	Functional class of the link (1=highway/throughway)
16	LONG	Toll of the link from B to A
17	LONG	Toll of the link from A to B

Table 5.2: *Input file format for TRANSIMS network links* — See table for details on columns.

column	format	description
1	LONG	Logical id of the node.
2	DOUBLE	x coordinate in Meters.
3	LONG	y coordinate in Meters.

Table 5.3: *Input file format for network nodes* — See table for details on columns.

Accessories (.acc)

This file contains the accessories of the street network. They serve as insertion or deletion point for vehicles. The format is described in Table 5.5.

Traffic Light Phases (.pha)

This file contains the green and red phases for traffic lights at intersections. The format is described in Table 5.6. The function of the the phases is described in 2.1.2.

Routeplan

This file contains the routes to be executed during simluation. In contrast to the network element input files it is **not** line oriented but contains tokens in a loose format. Each route is started with

column	format	description
1	LONG	Logical id of the link.
2	LONG	Logical id of the source node.
3	LONG	Logical id of the destination node.
4	LONG	Number of lanes (1..`MAX_LANES`).
5	DOUBLE	Length of the link given in Meters. This value is ignored. The length of the link is computed as the Euclidian distance between the nodes.
6	LONG	Class of the link. see (todo) The class of the link must be `1` for highways and a value other than `1` for all other types.
7	DOUBLE	Speed limit of the link given in Meters/Second. The maximum velocity of the CA vehicle is computed from this using define `REAL_VELOCITY_TO_CA_VELOCITY` in file `TrafficData/CAConfig.h`.
8	DOUBLE	Free speed of the link given in Meters/Second. This Value is used for the TRANSIMS link statistics if a link is (a) the number of vehicles on the link is larger than one and (b) the density is smaller than `TRANSIMS_LINK_STAT_GRIDLOCK_DENSITY=0.05`. See `TGridStat.C`.

Table 5.4: *Input file format for network links* — See table for details on columns.

column	format	description
1	LONG	Logical id of the link the accessory resides on.
2	LONG	Logical id of the destination node of the link.
3	LONG	Logical id of the accessory itself.
4	DOUBLE	Distance of the accessory from the source node of the link given in Meters. The width of the accessory is always `CA_DEFAULT_V_MAX` sites. Note that the location of an accessory may be shifted if it on a distributed link and at the time to close to the boundary.

Table 5.5: *Input file format for network accessories* — See table for details on columns.

the `ROUTE` token. It is followed by variable number of figures.

The format is described in Table 5.7.

column	format	description
1	LONG	Logical id of the node of the traffic light phase.
2	LONG	Logical id of the *incoming* link for which the phase is valid.
3	DOUBLE	Duration of the green phase in Seconds.
4	DOUBLE	Duration of the red phase in Seconds.

Table 5.6: *Input file format for traffic light phases* — See table for details on columns.

token	format	description
1	CHAR	Constant token `ROUTE`
2	LONG	Departure time step. Time of day in seconds.
3	LONG	Route plan flags. This is a bit coded field. `bit 0` is set if the source is an accessory. Otherwise it is a node. `bit 1` is set if the destination is an accessory. Otherwise it is a node. `bit 2` is set if the route can be shortened during transfer to another CPN.
4	LONG	Source id. Depending on the flags this is either an accessory id or a node id.
5	LONG	Destination id. Depending on the flags this is either an accessory id or a node id.
6	LONG	Logical route id.
7	LONG	Number of steps.
8	LONG	Logical link id of the next step.
9	LONG	Time estimate of the next step. The time is estimated for the *end* of the link.
...		
7+2*steps	LONG	Logical link id of the last step.
8+2*steps	LONG	Time estimate of the last step. The time is estimated for the *end* of the link.

Table 5.7: *Input file format for routes.* — See table for details on columns.

5.2.5 Computing

Link Timing Feedback (.edg.time)

This file contains information on the execution time of street links. If present and configured it will be used for the load balancing of the partitioning algorithm. The format is described in Table 5.8.

Node Timing Feedback (.nod.time)

This file contains information on the execution time of the intersections. If present and configured it will be used for the load balancing of the partitioning algorithm. The format is described in

column	format	description
1	LONG	Logical id of the link.
2	DOUBLE	Time in Seconds elapsed while processing the link logic.

Table 5.8: *Input file format for timing feedback for links.* — See table for details on columns.

Table 5.9.

column	format	description
1	LONG	Logical id of the node.
2	DOUBLE	Time in Seconds elapsed while processing the node logic.

Table 5.9: *Input file format for timing feedback for nodes.* — See table for details on columns.

5.3 Iterative Replanning

5.3.1 The Tasks of the Control Script

The simulation suite consists of three major applications and the control script `RunIterativeReplanning.py` which triggers the execution of the applications. It generates a configuration file from a template file which is adapted for each iteration.

In detail, the tasks of the control script for each selected iteration are as follows:

1. Prepare the execution of the planner by providing the TRANSIMS route file of the previous iteration. If the file had been archived it will be unzipped and moved to the `data` directory.

   Also, the link timing and node timing feedback files will be provided if available from the previous iteration.

2. Execute the `Planner` binary which takes a route file as input (in TRANSIMS format), re-routes a certain fraction of the routes contained therein, and writes a new route file (also in TRANSIMS format) to disk.

3. GNU-zip the TRANSIMS route file and the TRANSIMS link traveltime file of the previous iteration and move them to the `archive` directory.

4. Prepare the execution of the route converter by providing the TRANSIMS route file of the current iteration. It the file had been archived it will be unzipped and moved to the `data` directory.

5. Execute the `RouteConverter` binary which takes a TRANSIMS route file and clips the routes to the network which is used in the microsimulation. It converts the routeplans to the PAMINA format and writes them to disk.

6. GNU-zip the planner link hit statistics file (if produced) and move it the `archive` directory.

7. Prepare the execution of the microsimulation by providing the PAMINA route file of the current iteration and the node timing and link timing feedback files of the previous iteration. It the route file file had been archived it will be unzipped and moved to the `data` directory.

8. Execute microsimulation binary `ParSim` which takes read a plan file in PAMINA format and executes all routes contained therein.

9. GNU zip (if produced) the following files:

   - the PAMINA route file of the current iteration,
   - the completed routes statistics file,
   - the turn count statistics file,
   - the route by origin statistics file.

The script terminates if the requested number of iterations has been executed or any of the applications returns with a non-zero exit code.

Executing the Control Script

The script is located in the `bin` subdirectory of the Pamina home directory. It takes the following command line parameters:

`first-iteration=INDEX` Sets the index of the first iteration. Defaults to 1.

`last-iteration=INDEX` Sets the index of the last iteration. Defaults to 999.

`skip-planner` Skips the execution of the planner during the first iteration. Use this option if you know that the TRANSIMS route file already exists for the first iteration. Also see parameter `detect-restart-point`.

`skip-route-converter` Skips the execution of the planner during the first iteration. Use this option if you know that the PAMINA route file already exists for the first iteration. Also see parameter `detect-restart-point`.

`config-template=FILENAME` Sets the name of the configuration template from which the actual configuration named `Config` is produced.

`working-directory` Changes the working directory and all paths relative the working directory. If not set the directory in which the control script has been called will be used.
TODO/WARNING: this option has not been tested yet.

`detect-restart-point` Switch which prompts the control script to look for a good restart point within the first iteration. Depending on the data files that are available the execution of the planner and/or the route converter is skipped. The checks are as follows:

- If the TRANSIMS route file for the first iteration is found, the planner will be skipped.
- If the PAMINA route file for the first iteration is found, the route converter will be skipped.

`check-archive` Switch which prompts the control script to check the `archive` directory for GNU zipped files which are required for the current (usually the first) iteration. If a required file is found it will be unzipped and moved to the `data` directory.

`redirect-stdout` Switch which prompts the control script to redirect both standard outout and standard error of all applications to logging files in the `log` subdirectory. For each iteration there will be one file.

5.3.2 Cook Book

This section explains how to put all the input files and the configuration together and run a replanning scenario using the control script.

- Think of name for the scenario. In the following it will be referred to as *SCENARIO*.
- Create a directory which will be your work directory. This directory will be referred to as *WORK_DIR*. We recommend to make this directory a subdirectory of the `scenarios` directory.
- Create a subdirectory of the work directory which will contain most of the data files. The recommended name is `data`.
- Create a configuration file in the work directory from the template in the Braess paradox scenario. The file will be referred to as *CONFIG_FILE*.
- Check where the input files (eg. nodes and links) for the network representation are located and what the base file name is (e.g. the name without the extension).
- Check where the route file (TRANSIMS format) for the first iteration is. Make a link or a copy of the initial route file to the data directory. Use a certain naming scheme which includes the index of the iteration. For example: If the first iteration is number `1` a good name would be:

```
WORK_DIR/data/plans.TRANSIMS.000
```

The suffix `000` represents the iteration of the route file. The TRANSIMS file index is one lower than the iteration index since it is regarded as the output of the previous iteration. Also see the setting of `RouteConverter.Filename` below.

- Edit the configuration file and make required changes. For a list of likely candidates see 5.2.1.
- Double check all other setting in the configuration file.
- Make sure the parallel environment is up and running (see 5.1.2 for PVM and 5.1.3 for MPI).
- `cd` into the work directory.
- Start the control script:

```
<PAMINA_HOME>/bin/RunIterativeReplanning.py \
--first-iteration=FIRST_ITERATION \
--last-iteration=LAST_ITERATION \
--config-template=CONFIG_FILE
```

5.4 Output Files

5.4.1 Planner

The following files are generated by the planner.

Router Age Statistics (.age)

The file contains data about the age distribution of the routes. It is reported by class `RouteConverter/TRouteConverter.C`.

The file is started by three comment lines:

- `max age`: The maximum age of all routes (see columns 1 and 2 below).
- `average age`: The mean age of all routes.
- `not yet re-routed`: The fraction of all routes that have not been rerouted yet.

The format is described in Table 5.10.

Router Link Hits Statistics (.lhs)

The file contains the link hit statistics in TRANSIMS format. The statistics is stored in class `RouteConverter/TLinkHitStat` and reported by class `RouteConverter/TRouteConverter`.

column	format	description
1	LONG	Age of this route in routing iterations.
2	LONG	Maximum age of all routes minus the age of this route age.
3	LONG	Number of routes with this route age.
4	DOUBLE	Fraction of this route age.

Table 5.10: *Output file format for the route age statistics.* — See table for details on columns.

column	format	description
1	LONG	Logical id of the TRANSIMS link.
2	LONG	Logical id of the source node of the link.
3	LONG	Start of the time interval. The length of the intervals is defined by configuration parameter `LinkHitBinSize` in group `[RouteConverter]`.
4	LONG	Number of 'hits', e.g. the number of routes crossing this link in the given time interval.

Table 5.11: *Output file format for the TRANSIMS link hit statistics.* — See table for details on columns.

5.4.2 Route Converter

The following output files are produced by the route conversion utility which converts TRANSIMS routeplans into PAMINA format and trims them to the microsimulation network.

Router Conversion Logging (.log)

This file basically contains warnings about the conversion process. It is logged by class `TrafficData/TRouter`. There are no statistics whatsoever.

Typical messages are:

- `plan NNN is too short`: The number of TRANSIMS links is zero.
- `plan NNN does not touch map`: The route is completely outside the microsimulation network.
- `plan NNN has a single edge (not handled yet)`: The TRANSIMS route has its origin and destination on the same link. This is currently not supported by PAMINA.
- `no successor node found for NNN in plan NNN`: The TRANSIMS route contains an invalid node id which cannot be reached from the previous step of the routeplan.

- `inconsistent first edge in plan NNN`: The first link of the TRANSIMS plan is inconsistent.
- `inconsistent second edge in plan NNN`: The second link of the TRANSIMS plan is inconsistent.
- `plan NNN (time HH:MM) outside time window (HH:MM - HH:MM)`: The routeplan does not start inside the configured simulation interval.
- `no common initial node found for plan NNN`: There is no common node for the first two links of the TRANSIMS route.
- `cannot find source accessory NNN in plan NNN`: The source accessory of the TRANSIMS route does not exist in the PAMINA network.
- `cannot find destination accessory NNN in plan NNN`: The destination accessory of the TRANSIMS route does not exist in the PAMINA network.

Routing Statistics Logging (.cvs)

The file contains general data about the routing process. It is reported by class `RouteConverter/TRouteConverter`.

The format is described in Table 5.12.

column	format	description
1	LONG	End of departure interval in seconds.
2	DOUBLE	End of departure interval in hours. The fractional portion denotes minutes and seconds.
3	LONG	The number of routes with departure times in this interval.

Table 5.12: *Output file format for the routing statistics.* — See table for details on columns.

5.4.3 Simulation (CA and Route Execution)

Multilane CA Statistics (.mcs)

This file contains the basic statistics about vehicle velocities of the CA model. The first set of columns contains averages over all lanes. The next sets contain the statistics of specific lanes.

Note this important fact about the lane numbering: lanes are numbered from right to left, meaning that lane 0 ist always the *rightmost* lane.

The actual number of lanes reported depends on the maximum number of lanes found in all links of the traffic network. This may be lower than the maximum number of lanes `MAX_LANES` (see 4.2.3).

This statistics is stored by class `TrafficData/TMultilaneStat.C`, collected by class `TrafficData/TCAManager.C` and reported by class `ParSim/TParSimMaster.C`.

The format is described in Table 5.13.

column	format	description
1	LONG	Simulation timestep.
2	LONG	Number n_{veh} of all CA sites on all links and all lanes.
3	DOUBLE	Density of vehicles on all links and all lanes.
4	DOUBLE	Ratio of slow vehicles on all links and all lanes. A slow vehicle ist a vehicle that has a speed limit of less than `DEFAULT_V_MAX`.
5	LONG	Sum of CA velocities of all vehicles on all links and all lanes.
6	DOUBLE	Average CA velocity of all vehicles on all links and all lanes.
7	LONG	Standard deviation of CA velocities of all vehicles on all links and all lanes. $\sqrt{n_{veh} \sum v^2 - \frac{\sum^2 v}{n_{veh}}}$
8	LONG	Flow of all vehicles on all links and all lanes. $\frac{\sum v}{n_{Sites}}$
$i * 7 + 2$	LONG	Number $n_{site}(i)$ of all CA sites on lane i of all links.
$i * 7 + 3$	DOUBLE	Density of vehicles on lane i of all links.
$i * 7 + 4$	DOUBLE	Ratio of slow vehicles on lane i of all links. A slow vehicle ist a vehicle that has a speed limit of less than `DEFAULT_V_MAX`.
$i * 7 + 5$	LONG	Sum of CA velocities of all vehicles on lane i of all links.
$i * 7 + 6$	DOUBLE	Average CA velocity of all vehicles on lane i of all links.
$i * 7 + 7$	LONG	Standard deviation of CA velocities of all vehicles on lane i of all links. $\sqrt{n_{veh}(i) \sum v^2 - \frac{\sum^2 v}{n_{veh}(i)}}$
$i * 7 + 8$	LONG	Flow of all vehicles on lane i of all links. $\frac{\sum v}{n_{Sites}}$

Table 5.13: *Output file format for multilane CA statistics.* — See table for details on columns.

Turn Count Statistics (.tcs)

This file contains turn count statistics of intersections.

This statistics is stored in class `TrafficData/TTurnCountStat.C`, collected and reported by class `TrafficData/TTrafficSimulator.C`.

The format is described in Table 5.14.

column	format	description
1	LONG	Simulation timestep at which the statistics is written to file. The counts refer to the interval ending at this timestep!
2	LONG	Logical node id.
3	LONG	Logical link id of the incoming link.
4	LONG	Logical link id of the outgoing link.
5	LONG	Number of vehicle which were moved from the incoming link to the outgoing link during the statistics interval.

Table 5.14: *Output file format for intersection turn count statistics.* — See table for details on columns.

General Statistics (.rts)

The file contains general information about the process of the vehicle routing.

This statistics is stored in class `TrafficData/TRoutingStat.C`, collected and reported by class `TrafficData/TRouter.C`.

The format is described in Table 5.15.

Completed Routes (.crt)

This file contains one entry for each route that has been completed.

The data is collected and store in the route class itself `TrafficData/TRouteplan` and reported by class `TrafficData/TRouter.C`.

The format is described in Table 5.16.

TRANSIMS Link Travel Time Statistics (.tls)

The file contains the link travel time statistics in the original TRANSIMS format. PAMINA, however, does not collect travel times on links individually. Therefore, instead reporting individual travel times the router uses the current vehicle velocities to estimate the travel times on the links. If the number of vehicles on a links is less than `TRANSIMS_LINK_STAT_MIN_COUNT` the router assumes that the link is basically empty and reports an artificial traveltime computed from the free speed of the link. In this case the number of reported vehicles is set to 1.

The data is stored in class `TrafficData/TGridStat.c`, collected and reported by class `TrafficData/TRouter`.

The format is described in Table 5.17.

column	format	description
1	LONG	Simulation timestep at which the statistics is written to file. The counts refer to the interval ending at this timestep!
2	LONG	Application timestep (simulation wall clock time step).
3	DOUBLE	Application time (simulation wall clock time) in hours. The fractional portion corrsponds to minutes and seconds.
4	LONG	Number of vehicles on all links.
5	LONG	Number of vehicles (up to this timestep) that have been inserted *without* delay, that is *within* the time slack defined by `SP_NAME_SOURCETIMESLACK`.
6	LONG	Number of vehicles (up to this timestep) that have been inserted with delay larger than `SP_NAME_SOURCETIMESLACK`.
7	LONG	Number of vehicles (up to this timestep) that have been deleted (removed) from the simulation after a) having reached their destination or b) not having reached its destination ('failed vehicle').
8	LONG	Number of vehicles that are currently pending in sources waiting to be inserted into links.
9	LONG	Number of open routeplan files. This value usually corresponds to the number of active CPNs. During the final stages of a simulation (e.g. during the final few hundred time steps this number may decrease since some of the CPNs may have have read all routes from their respective files.
10	LONG	Number of vehicles (up to this time step) that have failed to reach the destination which was defined by their route plans. Note that this value is also included in column 7.

Table 5.15: *Output file format for the general routing statistics.* — See table for details on columns.

Routes By Origin Statistics (.rbo)

This file contains one entry for each route that has been completed and fulfills the following two requirements:

- The route did not start at an accessory. TODO: Why?
- The destination lies in a rectangle specified by the simulation configuration.

The data is collected and stored in the route class itself `TrafficData/TRouteplan` and reported by class `TrafficData/TRouter.C`.

column	format	description
1	LONG	Application timestep at which the the vehicle was scheduled to be inserted into the network. This figure is passed on from the original routeplan entry. The actual insertion time may have been later due to crowded sources. See also column 3.
2	LONG	Time in seconds that the router estimated for the completion of this route. This figure is computed from the original route plan entry as the difference between scheduled departure time and the arrival time of the last link of the route plan. See table 5.7. If re-routing is active this figure contains the travel time estimate for the most current re-routing.
3	LONG	Application timestep at which the vehicle was actually inserted into the network.
4	LONG	The time (in seconds) the vehicle had to wait before it was inserted into the network. It is computed as $col_3 - col_1$. This figure is always equal or greater zero.
5	LONG	The application time step at which the vehicle was deleted (removed) from the network.
6	LONG	The time (in seconds) that the vehicle spent in the network. It is computed as $col_5 - col_3$.
7	LONG	The delay (in seconds) that the route plan execution took longer in simulation that in the router estimate. It is computed as $col_6 - col_2$. Negative values denote faster route execution than estimated.
8	DOUBLE	The relative delay of the executed route. It is computed as col_7/col_2.
9	LONG	The route id. This figure is passed on from the original route plan entry.
10	LONG	The number of re-routings that the vehicle went through. This figure can only be larger than zero when the online re-routing is active.
11	LONG	Original estimated traveltime. This figure denotes the *first* travel time estimate before any re-routing was executed.

Table 5.16: *Output file format for the completed routes.* — See table for details on columns.

The format is described in Table 5.18.

column	format	description
1	LONG	Sum traveltime of all vehicles on the link.
2	LONG	Application timestep. The figures refer to the interval ending at this timestep!
3	LONG	Logical id of the node the link originates from.
4	LONG	Sum of the squares of the travel times of all vehicles on the link.
5	LONG	Number of vehicles on the link.
6	LONG	Logical id of the link.

Table 5.17: *Output file format for the TRANSIMS link statistics.* — See table for details on columns.

column	format	description
1	LONG	Application timestep at which the vehicle was actually inserted into the network.
2	LONG	The application time step at which the vehicle was deleted (removed) from the network.
3	LONG	The time (in seconds) that the vehicle spent in the network. It is computed as $col_2 - col_1$.
4	DOUBLE	X coordinate of the source given in Meters.
5	DOUBLE	Y coordinate of the source given in Meters.
6	DOUBLE	X coordinate of the destination given in Meters.
7	DOUBLE	Y coordinate of the destination given in Meters.
8	DOUBLE	Euclidian distance between source and destination in Meters.
9	DOUBLE	Average velocity of the vehicle during its route. This figure is computed as col_8/col_3.
10	DOUBLE	Angle $[-0.5 \ldots 0.5]$ between source location and destination location.
11	DOUBLE	Sector corresponding to the angle in column 10. The number of sectors is given by configuration parameter `RouteOriginSectors` in section `[Router]`. This figure is computed as $col_{10} * RouteOriginSectors$.

Table 5.18: *Output file format for the routes by origin statistics.* — See table for details on columns.

Rerouting Logging (.rrl)

The file contains data about the online re-routing process. It is reported by class `TrafficData/TRouter`.

The format is described in Table 5.19.

5.4.4 Computing

New Link Timing Feedback (.edg.time.new)

The file contains the link execution times of the simulation. It has the same format as the Link Timing Feedback. See table 5.8. The only difference is that this file is the output of the current simulation which will be used as input for the next simulation. The control script `RunIterativeReplanning.py` (see 5.3.1) will take care of renaming this file.

New Node Timing Feedback (.nod.time.new)

The file contains the node execution times of the simulation. It has the same format as the Node Timing Feedback. See table 5.9. The only difference is that this file is the output of the current simulation which will be used as input for the next simulation. The control script `RunIterativeReplanning.py` (see 5.3.1) will take care of renaming this file.

Performance Statistics (.perf)

The file contains data about the performance of the microsimulation. The statistics is collected and reported by `ParSim/TParSimMaster`.

The format is described in Table 5.20.

5.5 Graphics

The microsimulation provides a simple X windows graphical output. To use it you have to

- Activate the compilation of the X windows portions of the code by setting `XWINDOWS=1` in the file `config/ParSim.Settings`. This should be the default.
- Check your local site configuration file `config/`*SITE*`.Settings` for the correct path to your X windows include and library paths.
- If neccessary rebuild the applications using `make clean` and `make`. See 4.2.6.
- Activate the X windows of the microsimulation by setting `[ParSim].Graphics = 1;` in the respective configuration file.
- Make sure you have the `DISPLAY` variable of your environment set correctly.

As soon as the microsimulation starts a window will open showing the whole street network colored in CPN mode, in which the street segments are color coded by the CPN they reside on. For each CPN of the parallel machine a specific color is used.

In this window you can use the following keys:

2 Move the view one fourth of a window down.

8 Move the view one fourth of a window up.

4 Move the view one fourth of a window left.

6 Move the view one fourth of a window right.

– Zoom out by a factor of $\sqrt{2}$.

\+ Zoom in by a factor of $\sqrt{2}$.

m Switch to CPN mode in which the street segments are color coded by the CPN they reside on. For each CPN of the parallel machine a specific color is used.

v Switch to velocity mode in which the street segments are color coded by the average velocity of the vehicles.

d Switch to density mode in which the street segments are color coded by the density of the vehicles.

o Switch to object mode in which the location of the vehicles are individually displayed. Note that this mode generates a lot of network traffic and will slow down the simulation considerably.

q Close the view.

y Switch to delay mode in which the street segments are color coded by the average delay of the vehicles.

To open a new view just click on an existing view marking the lower left and upper right corners of the new view.

column	format	description
1	LONG	Type of the logging entry. Depending on the type some of the other columns may be missing. -1 Error while building the shortest path tree -2 Error while retrieving the shortest path result 0 The re-routing did not take place because the vehicle is to close to its destination. 1 Not used. 2 The re-routing was done as a full re-reouting, since all link travel times from the current location up to the destination were available. 3 The re-routing was done as a detour, since not all link travel times from the current location up to the destination were available. The detour is computed from the current link to the last link that still has travel times.
2	LONG	Number of links that were still be traversed before the re-routing was considered.
3	LONG	Number of links for which the travel time was estimated. This is equal to the number of links from the current location to the last link for which travel times are available.
4	LONG	Number of links replacing the re-routed portion of the travel plan.
5	LONG	Application time step of estimated arrival before the re-routing. If the vehicle has not been re-routed before this corresponds to the original planner estimate. Otherwise it is the estimate of the most previous re-routing.
6	LONG	Application time step of estimated arrival at the final destination after re-routing.
7	LONG	Application time step of the estimated arrival based upon the current link travel times if the vehicle continued to take its current route.
8	DOUBLE	Relative improvement of the re-routing which is computed as $col_7 - col_6$.
9	LONG	Result of the re-routing consideration: 1 Keep the route. No re-routing is attempted. 2 Route stays the same. Re-routing was attempted but no alternative route was found which would improve the remaining travel time. 3 A new route with a better remaining travel time was found.

Table 5.19: *Output file format for the online re-routing logging.* — See table for details on columns.

column	format	description
1	LONG	Simulation timestep.
2	LONG	Application timestep (simulation wall clock time step).
3	DOUBLE	Application time (simulation wall clock time).
4	LONG	Number of vehicles (objects) on on all links.
5	LONG	Wall clock time in seconds required for one simulation step.
6	DOUBLE	Real time ratio (RTR). This figure is computed as the number of simulation steps per wall clock seconds.
7	DOUBLE	Number of million CA site updates per wall clock second (MUPS).
8	DOUBLE	Number of million object updates per wall clock second (MOSPS).

Table 5.20: *Output file format for the simulation performance statistics.* — See table for details on columns.

List of Figures

List of Tables

Bibliography

[1] M. Van Aerde, B. Hellinga, M. Baker, and H. Rakha. INTEGRATION: An overview of traffic simulation features. *Transportation Research Records*, in press.

[2] R.K. Ahuja, Th.L. Magnanti, and J.B. Orlin. *Network Flows*. Prentice Hall, 1993.

[3] C.L. Barrett. personal communication.

[4] C.L. Barrett and M.A. Wolinsky. Incident-induced flow anomaly analysis and detection testbed final report: Design and concept demonstration. Technical report, Los Alamos National Laboratory, TSA-DO/SA, 1995.

[5] R.J. Beckman, K.A. Baggerly, and M.D. McKay. Creating synthetic baseline populations. *Transportation Research, A*, 30, #6:415–429, 1996.

[6] E. Ben-Naim, P. L. Krapivsky, and S. Redner. Kinetics of clustering in traffic flows. *Phys. Rev. E*, 50(2):822, 1994.

[7] G.D.B. Cameron and C.I.D. Duncan. PARAMICS – Parallel microscopic simulation of road traffic. *J. Supercomputing*, in press, 1996.

[8] CASim. URL http://rs1.comphys.uni-duisburg.de/OLSIM/.

[9] G.-L. Chang, H.S. Mahmassani, and R. Herman. Macroparticle traffic simulation model to investigate peak-period commuter decision dynamics. *Transp.Res.Rec.*, 1005:107–121.

[10] B.V. Cherkassky, A.V. Goldberg, and T. Radzik. Shortest path algorithms: theory and experimental evaluation. *Math.Prog.*, 73:129–174, 1996.

[11] Y.-L. Chou, H.E. Romeijn, and R.L. Smith. Approximating shortest paths in large-scale networks with an application to intelligent transportation systems. Technical report, University of Michigan, Ann Arbor, MI 48109, USA, January 1997. Report 95-21.

[12] A.J. Cuesta, F.C. Martínez, J.M. Molera, and A. Sánchez. Phase transitions in two-dimensional traffic flow models. *Phys. Rev. E*, 48(6):R4175–R4178, 1993.

[13] J. Esser. *Simulation von Stadtverkehr auf der Basis zellularer Automaten*. PhD thesis, University of Duisburg, Germany, 1997.

[14] J. Esser and M. Schreckenberg. Microscopic simulation of urban traffic based upon cellular automata. Technical report, Fachbereich 11 / Diskrete Mathematik, Gerhard-Mercator-Universität Duisburg, Germany, 1997. to be published in Int.J.Mod.Phys.C.

[15] M. Van Aerde et al. *INTEGRATION – Release 2, User's Guide*, 1995.

[16] J. Freund and T. Pöschel. A statistical approach to vehicular traffic. *Physica A*, 219(1–2), 1995.

[17] C. Gawron and P. Oertel. The PlayTraffic traffic simulation. Technical report, Traffic Group at the Center of Parallel Computing, University of Cologne, 1996.

[18] T.-Y. Hu and H.S. Mahmassani. Day-to-day evolution of network flows under real-time information and reactive signal control. *Transpn. Res.-C*, 5 No.1:51–69, 1997.

[19] S. Krauß, P. Wagner, and C. Gawron. A continuous limit of the Nagel-Schreckenberg model. *Phys. Rev. E*, 54:3707, 1996.

[20] H.S. Mahmassani, R. Jayakrishnan, and R. Herman. Network traffic flow theory: Microscopic simulation experiments on supercomputers. *Transpn. Res. A*, 24A (2):149, 1990.

[21] M. Marathe, D. Anson, M. Stein, M. Rickert, K. Nagel, and C.L. Barrett. Engineering the route planner for the Dallas case study. In preparation.

[22] T. Nagatani. Effect of traffic accident on jamming transition in traffic-flow model. *J. Phys. A*, 26(19):L1015, 1993.

[23] T. Nagatani. Bunching of cars in asymmetric exclusion models for freeway traffic. *Phys.Rev.E*, 51:992, 1995.

[24] K. Nagel. *High–speed Microsimulations of Traffic Flow*. PhD thesis, Universität zu Köln, 1994.

[25] K. Nagel. Individual adaptation in a path-based simulation of the freeway network of Northrhine-Westphalia. *Int.J.Mod.Phys.C*, 7 No.6:883–892, 1996.

[26] K. Nagel and C.L. Barrett. Using microsimulation feedback for trip adaptation for realistic traffic in Dallas. *Int.J.Mod.Phys.C*, 8 No.3:505, 1997. LA-UR 97-1334.

[27] K. Nagel, S. Rasmussen, and C.L. Barrett. Network-traffic as a self-organized critical phenomenon. In F. Schweitzer, editor, *International Conference "Self-organization of complex structures: From individual to collective dynamics"*, 1995.

[28] K. Nagel and A. Schleicher. Microscopic traffic modeling on parallel high performance computers. *Parallel Computing*, 20:125–146, 1994.

[29] K. Nagel and M. Schreckenberg. Traffic jam dynamics in stochastic cellular automata. In J.I. Soliman and D. Roller, editors, *Proceedings of the 28th International Symposium on Automotive Technology and Automation (ISATA)*, page 531, Croydon, England, 1995. Automotive Automation Ltd, Croydon, England. Paper No. 95ATS089, LA-UR 95-2132.

[30] K. Nagel, P. Stretz, M. Pieck, S. Leckey, R. Donelly, and C.L. Barrett. TRANSIMS traffic flow characteristics. Technical report, TSA-DO/SA, Los Alamos National Lab, 1997. LA-UR 97-3530.

[31] NRW-FVU. URL http://www.zpr.uni-koeln.de/Forschungsverbund-Verkehr-NRW/.

[32] United States Department of Transportation. URL http://www.dot.gov.

[33] PARAMICS. URL http://www.epcc.ed.ac.uk/epcc-projects/PARAMICS/.

[34] H.J. Payne. FREFLO: A macroscopic simulation model of freeway traffic. *Transportation Research Record 722*, 1979.

[35] J.T. Pfenning. *Beiträge zum Einsatz von "Workstation Clustern" als Parallel-Rechner.* PhD thesis, University of Cologne, 1994.

[36] 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 of Seminar D&E*, volume P 404-1, 1996.

[37] A.K. Rathi and A.J. Santiago. The new NETSIM simulation program.

[38] M. Rickert. Simulation zweispurigen Verkehrsflusses auf der Basis zellularer Automaten. Master's thesis, Universität zu Köln, 1994.

[39] M. Rickert. Parallel Toolbox 1.0. Technical report, Center for Parallel Computing, Cologne, Germany, and TSA-DO/SA, Los Alamos National Lab, USA, 1995.

[40] M. Rickert. *Traffic Simulation on Distributed Memory Computers.* PhD thesis, Center for Parallel Computing, University of Cologne, Germany, 1998.

[41] M. Rickert and K. Nagel. Experiences with a simplified microsimulation of the Dallas/Fort Worth area. *Int.J.Mod.Phys. C*, 8 No.3:483–503, 1997.

[42] M. Rickert, K. Nagel, M. Schreckenberg, and A. Latour. Two lane traffic simulations using cellular automata. *Physica A*, 231:534, 1996.

[43] M. Rickert and P. Wagner. Parallel real-time implementation of large-scale, route-plan-driven traffic simulation. *Int.J.Mod.Phys. C*, 7:133–153, 1996.

[44] H.E. Romeijn and R.L. Smith. Parallel algorithms for solving aggregated shortest path problems. Technical report, University of Michigan, Ann Arbor, MI 48109, USA, November 1997. Report 93-25.

[45] T. Schwerdtfeger. *Makroskopisches Simulationmodell für Schnellstraßennetze mit Berücksichtigung von Einzelfahrzeugen (DYNEMO).* PhD thesis, TH Karlsruhe, 1987.

[46] R. Smith, R. Beckman, D. Anson, and M. Williams. TRANSIMS, the TRansportation ANalysis and SIMulation System. Technical report, Los Alamos National Laboratory, 1995. LA-UR 95-1664.

[47] TRANSIMS. URL http://www-transims.tsasa.lanl.gov/.

[48] SMARTEST Project URL. http://www.its.leeds.ac.uk/smartest/.

[49] P. Wagner. Personal communication.

[50] R. Wimmershoff. URL http://www.rrz.uni-koeln.de/RRZK/Autoren/WF/traffic/OneLaneCA.html.

[51] K.E. Wunderlich, D.E. Kaufman, and R.L. Smith. Link travel time prediction techniques for convergent iterative anticipatory route guidance methods. Technical report, University of Michigan, Ann Arbor, MI 48109, USA, December 1997. Report 97-12.