Extension of a Multi-Agent Transport Simulation for Traffic Signal Control and Air Transport Systems

vorgelegt von
Dipl.-Inf.
Dominik Sebastian Grether
aus Offenburg

Von der Fakultät V – Verkehrs- und Maschinensysteme
der Technischen Universität Berlin
zur Erlangung des akademischen Grades

Doktor der Ingenieurwissenschaften
- Dr.–Ing. -

genehmigte Dissertation

Promotionsausschuss:

Vorsitzender: Prof. Dr. Thomas Richter
Gutachter: Prof. Dr. Kai Nagel
Gutachter: Prof. Dr. Peter Wagner

Tag der wissenschaftlichen Aussprache: 09. Januar 2014

Berlin 2014
D 83

Abstract

Goals

This thesis aims at the simulation based assessment of transport planning using a multi-agent simulation approach. Effects of transport policies like (de-)construction of infrastructure, changes in timetable, or regulations like speed limits or tolls can be analyzed with a high level of detail by the simulation under consideration[1]. Travelers are represented as individual entities that make their journey through the transport system and learn iteratively the modeled constraints. The thesis covers three areas: Traffic signal control, air transport systems, and software engineering.

Traffic Signal Control

It is shown how traffic signal control can be simulated with the multi-agent approach. Traffic flow is simulated by a computationally efficient queue model (Gawron, 1998b; Simon et al., 1999; Cetin, 2005). In this work, the queue model is extended to capture effects of traffic signals. A software component for microscopic modeling of traffic signals is developed. This component interacts with the traffic flow model and allows the simulation of network wide effects that result from a change of traffic signal control. Results indicate that the available choice dimensions of travelers, such as route choice or departure time choice, influence the evaluation of traffic signal control. The approach is applied to test different optimizations of traffic signal control. The optimization of offsets for coordination of adjacent junctions (green waves) has little impact on the network wide traffic patterns. In contrast, a traffic-actuated signal control results in network wide changes of travelers' route choice.

Air Transport Systems

The thesis also shows how the multi-agent approach is applied to air transport systems. The approach uses the public transport functionality of the simulation and modifies it for air transport. As a result, individual passengers are included into the modeling on all stages of their trip. Then, mode choice between air and alternative transport modes is added. The existing Multinomial Logit model for mode choice (Rieser et al., 2009; Rieser, 2010) is enriched by a Path Size Logit formulation that takes path overlap into account. This removes artefacts of the sampling process and enables the analysis of competitive markets, e.g., between high speed rail and air transport.

Software Engineering

The software architecture of the simulation was initially a monolithic piece of software, difficult to customize, and appeared not suited for further research. The thesis discusses the redesign of the software. Design goals aim at a modular, extensible architecture that permits researchers to modify or add certain components to the overall simulation under the assumption that suitable interfaces are available. As proof of concept, the module for fixed-time traffic signal control is provided as extension. The module is decoupled from the overall simulation approach and can be replaced in part, or completely.

[1] MATSim, see `www.matsim.org`

Zusammenfassung

Ziele der Arbeit

Die vorliegende Arbeit widmet sich der simulationsbasierten Verkehrsplanung anhand eines Multi-Agenten-Ansatzes. Verkehrsplanerische Maßnahmen, wie Rück- oder Neubau von Infrastruktur, Änderungen in Fahrplänen, Maut oder Geschwindigkeitsbegrenzungen, können anhand der verwendeten Simulation[2] mit einer hohen Detailgenauigkeit untersucht werden. Der Fokus liegt auf dem Verkehrsteilnehmer, der sich in der Simulation mikroskopisch durch das Verkehrssystem bewegt und in einem iterativen Prozess dessen Rahmenbedingungen lernt. Die Arbeit behandelt die Themen Lichtsignalanlagensteuerung, Luftverkehr und Softwareentwicklung.

Lichtsignalanlagensteuerung

Die Arbeit zeigt, wie Lichtsignalanlagen (LSA) in der Simulation abgebildet werden können. Der Verkehrsfluss wird durch ein sehr effizient zu berechnendes „Queue Model“ (Gawron, 1998b; Simon et al., 1999; Cetin, 2005) abgebildet. Anhand einer Erweiterung des „Queue Model's“ können die Effekte von LSA abgebildet werden. Ein entwickeltes Software-Modul modelliert mikroskopische LSA. Durch die Interaktion dieses Moduls mit dem Verkehrsflussmodell können Reaktionen der Nutzer auf eine Änderung der LSA-Steuerung netzwerkweit simuliert werden. Die Resultate zeigen, dass die modellierten Wahlmöglichkeiten der Reisenden, wie z.B. Abfahrtszeit- oder Routenwahl, die Evaluation von LSA-Steuerungen beeinflussen. Verschiedene Optimierungen der LSA-Steuerung werden evaluiert. Die Optimierung von grünen Wellen hat nur geringen Einfluss auf die Verkehrsmuster. Dahingegen kommt es bei einer verkehrsabhängigen Steuerung zu netzwerkweiten Änderungen der Routen.

Luftverkehr

Die Arbeit zeigt weiterhin, wie der Multi-Agenten-Ansatz zur Abbildung von Luftverkehr eingesetzt werden kann. Dabei wird das Simulationsmodul für öffentlichen Nahverkehr genutzt und entsprechend angepasst. Somit werden Passagiere auf allen Teilen ihrer Reise personenscharf abgebildet. Daraufhin wird die Verkehrsmittelwahl in die Modellierung aufgenommen. Das existierende multinomiale Logit-Modell für die Verkehrsmittelwahl (Rieser et al., 2009; Rieser, 2010) wird durch eine „Path Size Logit“-Formulierung erweitert, die Überlappungen von Routen explizit berücksichtigt. Dies behebt Artefakte des Sampling-Prozesses und ermöglicht somit die Analyse von Wettbewerbsmärkten zwischen Verkehrsträgern, z.B. zwischen Hochgeschwindigkeitszügen und Flugverbindungen.

Softwareentwicklung

Die Software-Architektur der Simulation war anfangs monolytisch aufgebaut, funktionell schwer zu erweitern und schien für weitere Forschungszwecke nicht geeignet. In der Arbeit wird das Redesign auf eine modulare, erweiterbare Architektur erläutert. Diese soll es Forschern erlauben eigene Softwarekomponenten als Erweiterung bereitzustellen. Wie am Beispiel des Moduls für LSA gezeigt wird, ist dies möglich, sofern geeignete Schnittstellen zur Verfügung gestellt werden. Das Modul ist vom eigentlichen Simulationsprozess weitgehend abgekoppelt und kann in Teilen oder komplett ersetzt werden.

[2] MATSim, `www.matsim.org`

Contents

Chapter 1

Introduction

1.1. Motivation

Transport Planning

Running transport systems efficiently can ease life of travelers within the system and also may have impacts on economy, environment, and society as a whole. Transport planning tries to improve efficiency whereby the understanding of "efficient" varies. Planning always implies some kind of forecasting that is uncertain by nature. To predict the impacts of transport policies, as (de-)construction of infrastructure, changes in timetables, or regulations like speed limits or tolls, this work uses an agent-based simulation approach. Transport policies are often costly and financed by public money, so one may ask if an agent-based simulator is the appropriate tool for appraisal.

Multi-Agent Simulation

Before we can address this question, we have to clarify the semantics for the overloaded "*agent*" paradigm (Petrie, 2007). In our context, "*multi-agent*" originally denotes the modeling of each traveler in the transport system as individual entity throughout the entire simulation process. The behavior of travelers is represented by more theoretical attempts to characterize and forecast travel behavior (e.g. Nagel and Flötteröd, 2012). The project has a long and outstanding history, there are too many publications to cite them at this point[1]. Thus, the approach is not considered to be *"alchemy"* (Petrie, 2007). Forecasting, however, always comes with a taste of alchemy.

Compared to more traditional transportation planning approaches as the four-step process, the modeling of transport systems with a multi-agent simulator has advantages and disadvantages. The former includes a plausible modelling of travelers' choice of transport mode while it comes with a valid interpretation (Rieser et al., 2009). Also, for economic appraisal of transport policies the agent-based approach appears well suited (e.g. Nagel et al., 2008; Kickhöfer et al., 2011; Kaddoura et al., submitted). The

[1] The reader is referred to `www.ivt.ethz.ch/docs/index` and `www.vsp.tu-berlin.de/publications/`, last access 27.11.2013, for further publications.

agent-based modeling seems more intuitive and easier to explain than traditional approaches for many problems. Since the simulation, that is developed and used in this work, can be applied to large-scale problems, e.g., whole metropolitan areas, network wide effects of a policy can be studied. In principle, this is not limited to a single policy. The effects of several policies can be analyzed jointly. But an agent-based simulator is still a piece of software. It inherits all drawbacks and failures common to software.

Research & Software

Commercial software applications are considered as useful tools if they solve the problem of interest. For research, the use of software is more critical. In a recent article, Joppa et al. (2013) point out several problems if research is based on software tools. The existence of a specific software that is already successfully used in peer reviewed publications can blur scientific decisions. The choice of a software based method is often motivated by easy use or existing publications instead of a validation against other methods. While papers based on research software are typically peer reviewed, the underlying software is not reviewed at all. Often, presented results lack reproducibility and transparency. This can be improved by use of open source software (Hatton, 2007). Thus, Hatton (2007); Joppa et al. (2013) argument for peer reviewed code in alignment with publications.

Computer Science & Software Engineering

Neither computer science, nor software engineering can help researchers to select their methodology and tools carefully. In general, software engineering is seen and taught as area in computer science. Following Joppa et al. (2013); Offutt (2013), this might be subject to change. Software engineering should be a *"core part of the science curriculum"* not restricted to studies of computer science (Joppa et al., 2013).

Scientific Software Engineering

Scientific findings from computer science can help when developing scientific software. But, it may take decades before they can be applied in software engineering. The most prominent example might be the rise of the "world wide web" over the last decades and the recent "web 2.0" software engineering approaches that ease development of www applications. Approaches for software engineering mostly focus on the development of commercial software. Science is not adopting technologies from software engineering, as methodology for commercial software is not suited for scientific application development (Kelly, 2007; Sanders and Kelly, 2008; Carver et al., 2013; De Roure and Goble, 2009). Despite the lack of methodology, there is some empirical evidence why scientific software development is different to the commercial counterpart. Kelly et al. (2009) argue for a separation of data, computational functions, and user interfaces. The overall approach should stick to standards rather then to customized solutions. De Roure and Goble (2009) care about scientific workflows and make a strong argument for reusable software that is suited for extensions and customizations. Closed-source code results in a *black box*. Use of a black box increases the risk for biased or wrong deductions from the output of the software (Sanders and Kelly, 2008).

Developing Scientific Software

At least in the transportation context, the argument made by Carver et al. (2013), that most of the computational science and engineering developers *"don't know what they*

don't know", is twofold. Knowledge and understanding of computer science and software engineering comes with a lack of experience in transport related problems. To bridge the gap between both worlds, it seems reasonable, to work on both sides. Therefore, this work covers two extensions to the multi-agent simulation approach — for the simulation of traffic signal control and air transport systems.

Traffic Signal Control

Traffic signals ensure security of travelers at junctions and regulate right of way. Furthermore, by assigning green times to the different approaches of a junction they are a determinant of the junctions performance. Fixed-time traffic signal control repeats periodically the same schedule for signalization. Traffic-responsive signal control reacts dynamically on the prevailing traffic patterns to improve the performance of the junction or the system as a whole. Even if traffic-responsive control improves the traffic conditions at a single junction, it might not result in benefits for the system as a whole. As result of an improved, traffic-responsive signal control at two single junctions, network wide changes in travel patterns can evolve Burghout and Wahlstedt (2007). Hu and Mahmassani (1997) argue that second order or network effects should be taken into account when effects of signal control strategies are tested. Network effects include drivers' reactions not only in terms of route choice but also in terms of scheduling. Traffic-responsive signals need to obey some constraints. Otherwise, traffic may become unstable at the network level. Thus, traffic-responsive signals can perform much worse than a fixed-time control in some situations (Lämmer and Helbing, 2010). The simulation can capture most of these effects. Thus, it is well worth to consider a extension for the simulation of traffic-responsive signal control. Further, the impacts of recently developed optimization models for fixed-time control can be tested (Köhler and Strehler, 2010).

Simulation Tools

The effectiveness of traffic signal control can be simulated on a high level of detail, e.g., with the commercially available simulation tools VISSIM (PTV AG, 2008) or Aimsun (Barceló et al., 2005). Because of their commercial nature, these models are closed-source, i.e., a black box. Furthermore, non commercial research tools as SUMO[2] (e.g. Krajzewicz et al., 2005) or ITSUMO[3] (Bazzan et al., 2010a, 2011) are available. These tools implement explicit car following and lane changing models. Common to all these models is that the demand is given as origin-destination (O-D) matrices, and their capability to simulate large-scale scenarios. Also common to all, however, is a comparatively high computational effort for one simulation run. The traffic flow model used in this work (Gawron, 1998b; Simon et al., 1999; Cetin et al., 2003; Cetin, 2005) has less details and thus saves computation time. This can be important if travelers' reactions on changes of signal control on large-scale networks shall be analyzed.

Air Transport Systems

For the simulation of air transport systems, many simulation approaches are available (e.g. Bilimoria et al., 2000; Sweet et al., 2002; Alam et al., 2008; Clarke et al., 2007). Most of them aim at the technology of air transport systems. The model proposed in

[2] see `sumo-sim.org/`, last access 07.10.2013

[3] see `wiki.inf.ufrgs.br/ITSUMO_DOC_en`, last access 07.10.2013

this work represents technology rather coarse, but similar to the approach by Clarke et al. (2007). In contrast, a high detail resolution of passengers is available at all stages of their trips and is not restricted to air transport. At least in a European context, competition between air, rail, and car transport exists. The proposed approach can analyze passengers' reactions between different transport modes.

1.2. Outline, Contributions & Limitations

In the subsequent chapter, the currently applied methodology and technology of the multi-agent transport simulation approach is introduced. More detailed motivation, relevant backgrounds, and literature is then given in the respective chapters.

Traffic Signals

Chapter 3 considers options to represent traffic signals within the simulation approach. Several approaches to model dynamics of transport networks are reviewed. Most suited appears an extension of the traffic flow model. The default model for traffic flow of the simulation, a queue model (Gawron, 1998b; Simon et al., 1999; Cetin et al., 2003; Cetin, 2005), is then extended to capture effects of traffic signals. Other traffic flow models are not considered. The simulation of traffic signals may have impacts on the calculation of routes. These are captured by a time dependent shortest path algorithm based on Dijkstra (1959). Other options for calculation of shortest paths remain out of consideration.

Software Engineering

Chapter 4 explains important aspects of the software engineering and design for the simulation and highlights options for extension and customization. The chapter makes use of standard solutions for object-oriented software and explains, why certain approaches are chosen or neglected. The focus is on the programming language Java. The spelling style should be understandable after some basic training in object-oriented programming. E.g., after attending the one term, 4 h lecture that is part of our engineering curriculum at TU Berlin. Neither lambdas and closures, nor functional programming is covered by the lecture. The provided explanations do not rely on these more advanced concepts. The chapter shows by example, how long term stable, decoupled extensions can be provided to the simulation. As example, an extension for traffic signal control is presented.

Network Effects, Traffic Signals

Chapter 5 studies network wide effects of traffic signal control. On a small example network, potential impacts of traffic-responsive control are analyzed. Then, a simulation scenario for a real-world instance is set up. The scenario serves as base for further analysis of network wide effects due to changes in traffic signal control. The results show that the chosen approach can capture such effects, but they are not advanced enough to compare and assess the two traffic signal control strategies under consideration. The project has started February, 2013.

Air Transport Systems

Chapter 6 explains, how air transport systems can be simulated with the multi-agent approach. The technology side of air transport systems is modeled on a low level of

detail. The focus of attention is not on air traffic control. It is shown, however, how influences of air traffic control, taxiing, or weather conditions could be captured despite the model's low level of detail. Then, results for a simulation of the German air transport demand on a Europe to world wide air technology model are presented. Air line choice and pricing are not covered by the model, but could be included in further studies.

Finally, the thesis ends with a conclusion revisiting arguments from the motivation.

Please note that parts of this thesis have been presented at conferences and are already published in conference proceedings and journals (Grether et al., 2009a; Kickhöfer et al., 2011; Grether et al., 2011, 2012, 2013; Grether and Nagel, 2013b,a). A more detailed review of reused material is given in the introduction of the relevant chapters.

Chapter 2

Multi-Agent Transport Simulation

Introduction

The simulation approach used in this thesis is based on the software tool MATSim[1]. The next paragraphs provide an overview of the simulation approach and highlight the most important details used in this work. For more detailed information on technical aspects, please see Raney and Nagel (2006); Balmer et al. (2005b). For a detailed discussion of methodology the reader is referred to Nagel and Flötteröd (2012). Regarding economic concepts used in the simulation approach, see Nagel et al. (2008); Kickhöfer et al. (2011).

Please note that this chapter reuses and extends in part Grether et al. (2009a); Kickhöfer et al. (2011).

2.1. Simulation Overview

Iterations

In MATSim, each traveler of the real system is modeled as an individual virtual person[2]. The approach consists of an iterative loop that has the following important steps (Fig. 2.1):

1. *Plans generation:* All virtual persons independently generate daily *plans* that encode, among other things, their desired activities during a typical day as well as the transportation mode for each leg between activities. Virtual persons typically have more than one plan ("plan database").

2. *Mobility Simulation:* All selected plans are simultaneously executed in a simulation of the physical system (often called "network loading" or "traffic flow simulation").

[1] Multi-Agent Transport Simulation `www.matsim.org`.
[2] In other works travelers are often referred as "agent"

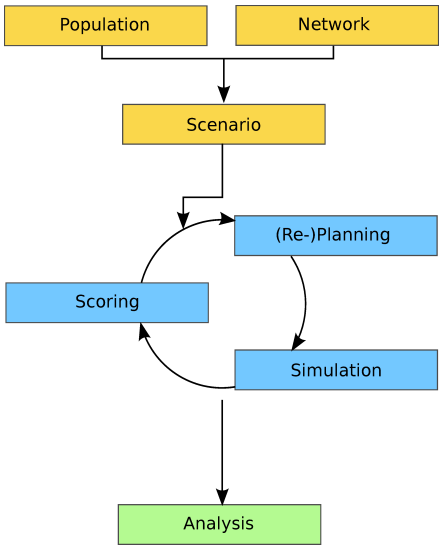


Figure 2.1.: MATSim simulation process, overview (source: Own figure (Grether et al., 2007), modifications can be found at many places)

3. *Scoring:* All executed plans are scored by an *utility function* which can be personalized for every individual.

4. *Learning/(Re-)Planning:* At the beginning of every iteration, some virtual persons obtain new plans by modifying copies of existing plans. This is done by several *modules* that correspond to the choice dimensions available, e.g., time choice, route choice, and mode choice. Virtual persons choose between their plans according to a Random Utility Model (RUM). The number of plans per virtual person is limited because of memory constraints and is typically set to 4 or 5. If this threshold is exceeded, the plan with the lowest score is deleted.

Learning

The repetition of the iteration cycle coupled with the plan database enables the virtual persons to improve (learn) their plans over many iterations. This is why it is also called *learning mechanism*. The iteration cycle continues until the system has reached a relaxed state. At this point, there is no quantitative measure of when the system is "relaxed"; we just allow the cycle to continue until the outcome is stable.

Interpretation

In the steady state, the model is equivalent to the standard multinomial logit model (Nagel and Flötteröd, 2012)

$$p_j = \frac{e^{\mu \cdot V_j}}{\sum_i e^{\mu \cdot V_i}} \,, \tag{2.1}$$

where p_j is the probability for plan j to be selected and μ is a sensitivity parameter, set to 2 for the simulations in this work. In consequence, V corresponds to the systematic component of utility in Random Utility Models (RUM) (e.g. Ben-Akiva and Lerman, 1985; Train, 2003), where utility is defined as $U = V + \epsilon$. In RUM, the ϵ is called random

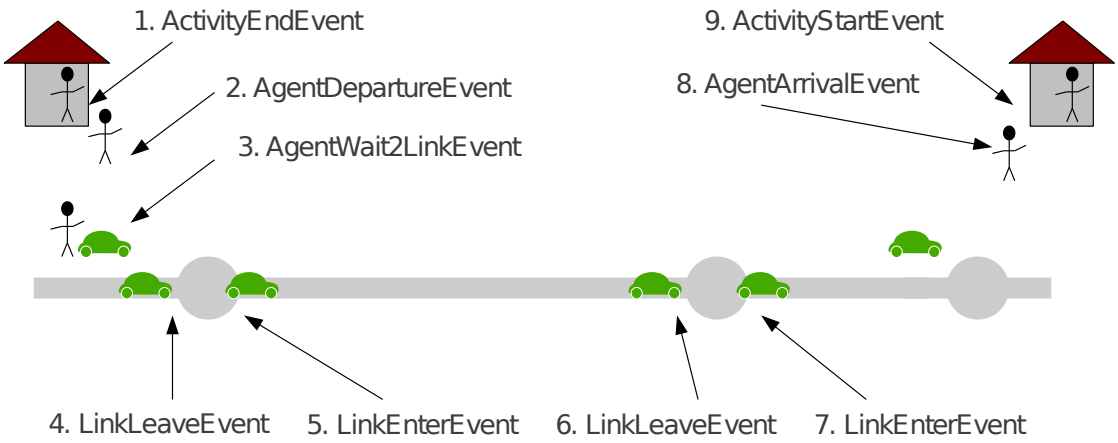


Figure 2.2.: Sequence of events for a single virtual person on its trip between two activities (source: Own figure, serves as base for official documentation `matsim.org/node/598`, last access 18.11.2013)

component of utility, and in order to arrive at equation (2.1), epsilon is assumed to be Gumbel distributed.

2.2. Mobility Simulation

Overview

The mobility simulation consists of a model of the physical environment, a model of traffic flow, and several agent-representations. The physical environment comprises at least a model of the transportation network. Agent-representations exist for virtual persons, drivers of public transit vehicles, etc.

Traffic Flow Model

The traffic flow model is a queue model, that moves vehicles through the transportation network. Queue models for traffic flow disregard most of the details of vehicle movements on a road (Gawron, 1998b; Simon et al., 1999; Cetin et al., 2003; Cetin, 2005). Further details of the traffic flow model are explained in Sec. 3.2.

Events

The output of the traffic flow simulation is a list of "*events*" for each vehicle/virtual person, such as entering/leaving link, left/arrived at activity, and so on, see Fig. 2.2. Data for an event includes which vehicle/virtual person experienced it, what happened, at what time it happened, and where the event occurred. With this data it is easy to produce different kinds of information and indicators like link travel time (which, e.g., will be used by the router), trip travel time, trip length, percentage of congestion, and so on.

2.3. Scoring

Utility Function

In order to measure the quality of a plan after execution and to compare plans, it is necessary to assign a quantitative score to the performance of each plan. For this purpose the utility function of the virtual persons is used. The total utility $[utils]$ of a plan is computed as the sum of individual contributions:

$$V_{total} = \sum_{i=1}^{n} V_{perf,i} + \sum_{j=1}^{n} V_{tr,j}\,, \tag{2.2}$$

where V_{total} is the total utility for a given plan; n is the number of activities, which equals the number of trips (the first and the last activity are counted as one); $V_{perf,i}$ is the (positive) utility earned for performing activity i; and $V_{tr,j}$ is the (usually negative) utility earned for travelling during trip j.

Activities

For calculation of $V_{perf,i}$ a logarithmic form is used

$$V_{perf,i}(t_{perf,i}) = \beta_{perf} \cdot t_{*,i} \cdot \ln\left(\frac{t_{perf,i}}{t_{0,i}}\right)\,, \tag{2.3}$$

where t_{perf} is the actual performed duration of the activity, t_* is the "typical" duration of an activity, $t_{0,i} = t_{*,i} \cdot e^{-10/t_{*,i}}$ a scale parameter, and β_{perf} is the marginal utility of an activity at its typical duration. β_{perf} is the same for all activities, since in equilibrium all activities at their typical duration need to have the same marginal utility. In this work, a β_{perf} of $6/h$ utils is used.

Traveling

The (dis)utility of traveling is linear in travel time, i.e.,

$$V_{tr,j}(t_{tr,j}) = \beta_{tr} \cdot t_{tr,j}\,, \tag{2.4}$$

whereby $t_{tr,j}$ is the experienced travel time on trip j, and β_{tr} the marginal utility of traveling. In this work, β_{tr} is set to $-6/h$ for all virtual persons. If survey data is available, β_{tr} can be estimated for each virtual person and transport mode (Kickhöfer et al., 2011). If a virtual person arrives too late at his "work" activity, a penalty of $\beta_{late} \cdot t_{late}$ is added to the overall utility. In this work, β_{late} is set to $-18/h$.

Further details on the default MATSim utility function can be found in Charypar and Nagel (2005), for an interpretation of the parameter values the reader is refered to Nagel et al. (2008). Kickhöfer et al. (2011) contains one of the most recent discussions of this utility based approach.

2.4. Re-Planning

Choice Dimensions

The Re-Planning comprises several modules that can be en- or disabled according to the

choice dimensions available to virtual persons. Note that the modules in this section describe how new alternatives are generated, i.e., how the choice sets of the agents are extended over the iterations. Actual choice is made according to Eq. (2.1) within those options that an agent has memorized.

2.4.1. New Routes

Shortest Paths

The generation of new routes for the "car" mode trips is implemented based on a time dependent shortest path algorithm. Apart from relatively small and essential technical details, the implementation of such an algorithm is straightforward (Jacob et al., 1999; Lefebvre and Balmer, 2007). The shortest path algorithm calculates link travel times from the event output of previous mobility simulations. The travel times of each link in the transport network are encoded in time bins (default $15\,min$). They serve as input for the weight function that calculates generalized costs. The least generalized cost path from each activity to the next one is calculated as function of departure time. As default shortest path algorithm, a time dependent implementation of Dijkstra's algorithm (Dijkstra, 1959) is used.

2.4.2. New Time Structures

Random Time Allocation

Modification of the time structure is implemented by a simple approach that applies a random "mutation" to the duration and/or the end time attribute of a virtual person's activities in a plan. For each such attribute, a random time from the uniform distribution $[-2\,h, +2\,h]$ is selected and added to the attribute (configurable). Any negative duration is reset to zero.

2.4.3. New Transport Modes

Random Transport Modes

New transport modes are also generated by a random approach. For each leg between two activities in a virtual person's plan, one of the available transport modes is selected randomly. Again a uniform distribution is used; for a quantitative interpretation see (Rieser et al., 2009) and Chapter 6.

2.5. Public Transit

Overview

The public transit module of MATSim aims at the microscopic simulation of public transit, concentrating on several types of ground transportation, e.g., buses, streetcars or para transit (Rieser, 2010). This approach is successfully applied in ground transportation planning (e.g. Neumann et al., 2014).

Transit Schedule

In a transit schedule, transit stop facilities, lines, and routes are specified. Passengers can access and leave vehicles at transit stops. Each transit line contains one or more transit routes. Transit routes specify the order in which stops are lined up to a route and the departure time of a vehicle at the beginning of the route. Furthermore each route specifies which links in the network are used to connect stop facilites.

Vehicles

Characteristics of transit vehicles are specified using the default configuration of the MATSim framework[3]. Several *vehicle types* can be defined that contain information as length, width, passenger capacity, maximum velocity, and energy consumption. The vehicle type specifies how fast passengers can access and leave a vehicle. In addition to the different vehicle types, a set of particular vehicles can be defined. Each vehicle has exactly one type assigned and inherits all attributes. The individual vehicles are inserted into the traffic flow simulation and moved by the queue model along their routes.

Passenger Routing

Virtual persons, that travel on the microsimulated public transit mode, require some encoding of their route, i.e., a specification of the transit stops for boarding and alighting, the transit line and route, and the transfers between different lines. With the information in the transit schedule, a graph is set up. This graph serves as input for the time dependent shortest path algorithm. In contrast to trips on mode "car", in the current implementation shortest path calculations do not consider experienced travel times or overcrowded vehicles. The weight function makes use of the scheduled travel times. On edges that model transfers, an additional, homogeneous penalty can be added to the travel time ($c_{lineswitch}$). For further details of passenger routing in the public transit module, the reader is referred to Rieser (2010).

[3] `http://matsim.org/files/dtd/vehicleDefinitions_v1.0.xsd`

Chapter 3

Extensions for Traffic Signal Control

Overview

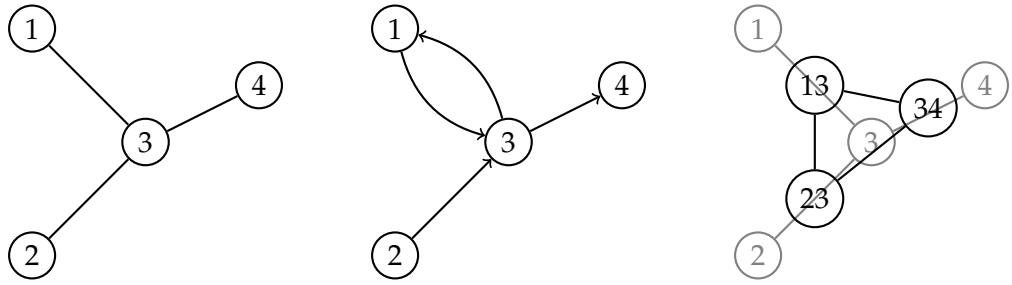
This chapter considers options for a computationally affordable, but microscopic, simulation of traffic signal control and its integration into the simulation framework. As the simulation is explicitly designed for large-scale applications, computational aspects, i.e., run time and memory constraints, are considered. Dependent on the chosen solution, the integration affects the network representation, the traffic flow model, and the calculation of shortest paths. Shortest paths shall be calculated by a time dependent implementation of Dijkstra's algorithm (Dijkstra, 1959). For the representation of dynamic networks several approaches exist (Ford and Fulkerson, 1962; Pallottino and Scutella, 1998; Köhler et al., 2009; George and Shekhar, 2008). For MATSim, a model that captures time variant attributes is available (Lämmel et al., 2010). The chapter discusses why all these approaches are not suited for large-scale applications. Traffic signal control is thus modeled by a queue model for traffic flow. Other traffic flow models are not considered further. Finally, an approach based on Gawron (1998b); Simon et al. (1999); Cetin et al. (2003); Cetin (2005) is selected.

Please note that Sec. 3.2 of this chapter provides an extended version of Grether et al. (2012) and reuses in part material from the paper.

3.1. Network Representation, Graph Theory & Semantics

Network Representation

Most problems in the transport planning domain require a representation of the transport network. For real-world problems this representation of the transport network is derived from a wide range of data, e.g., using satellite images (Birkmann et al., 2008) or road map data (Chen et al., 2008), and is then persisted in some data format. The representation of the data format, however, might not fit to the representation required by a transport model. Frequently, transport models use a *graph* to represent the transport network. In conjunction with other data that describe the transport network, the notion of graphs quickly gets inaccurate. In theory, graphs and networks are synonym.



(a) Graph (b) Directed graph (c) Line graph of the graph in (a)

Figure 3.1.: Different types of graphs

In practice, the term transport network often has a more general semantics. It seems worth, to look at the subtle differences in more detail.

Graph Theory

Graph theory provides problem formulations, proofs, and algorithms that are build on graphs as abstract structure. For formal definitions the reader is referred to (Diestel, 2010). A good introduction focusing on transport related problems can be found in (Schultes, 2008; Geisberger, 2011). In the following, only the concepts required for the further arguments in this work are shortly introduced and explained on small examples.

Graph

A graph G is a pair $G = (V, E)$ that consists of a set of *vertices* V and a set of *edges* E. A vertex models an abstract object. Edges represent a pairwise relation between two vertices. Fig. 3.1a shows a typical graphical representation of a graph with $V = \{1, 2, 3, 4\}$ and $E = \{\{1, 3\}, \{2, 3\}, \{3, 4\}\}$. Each vertex has a label to distinguish vertices and specify edges, i.e., the relations between the objects represented by vertices. So far, there is no more information attached to the graph. Particularly, coordinates for vertices are not included in the definition of a graph. A graph could be drawn in many orientations. In principle, additional information can be added to vertices and edges via functions. Frequently, a weight or cost function $w : E \to \mathbb{R}$ is defined, representing costs or weights of each edge.

Directed Graph

Line Graph

Graph theory distinguishes between many types of graphs. For our purposes, *directed* graphs and *line* graphs are of interest. In a directed graph, relations have a direction. A directed graph is shown in Fig. 3.1b. The relation $\{1, 3\}$ is bidirectional, while the relations $\{2, 3\}$ and $\{3, 4\}$ are unidirectional. A line graph $L(G)$ is a graph that is derived from another graph G by an "inversion" of the structure. Fig. 3.1c shows the line graph of the graph in Fig. 3.1a. Each edge of G is converted to a vertex of $L(G)$. An edge is added between two vertices a, b of the line graph $L(G)$ if the edges in G that are used to derive the vertices a, b are connected by a node in G.

Transport Networks

Transport networks are often represented as graph. Well known examples are subway layouts of urban areas where each vertex depicts a subway station and each edge

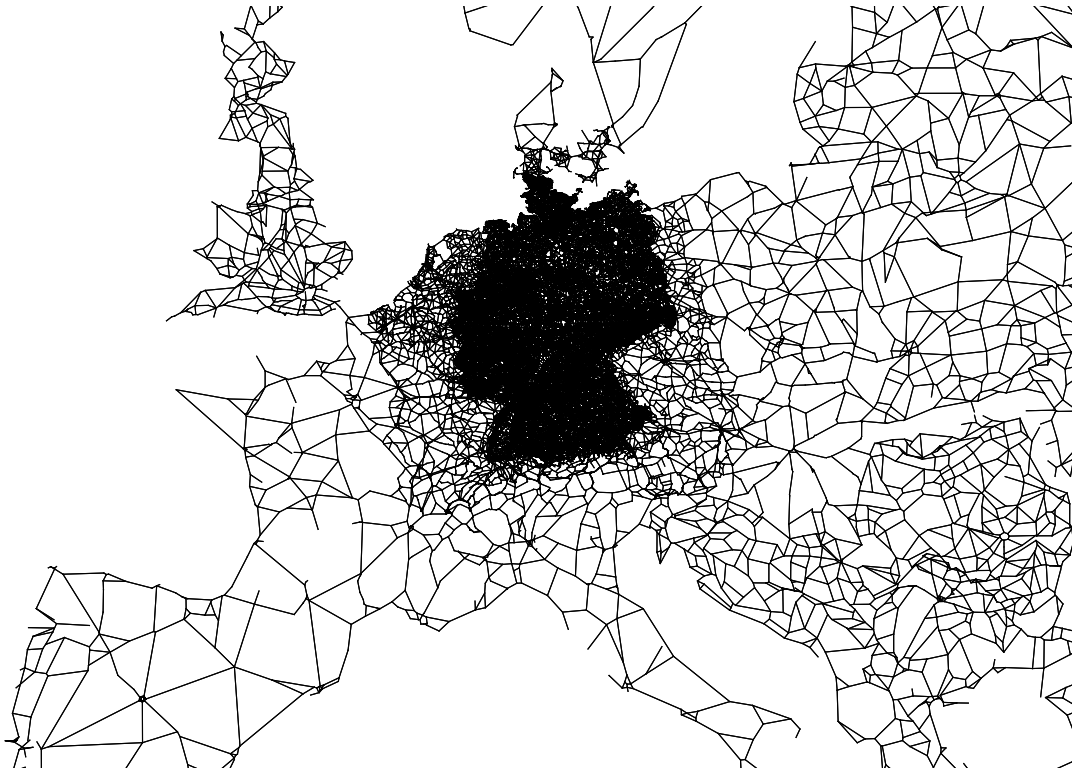
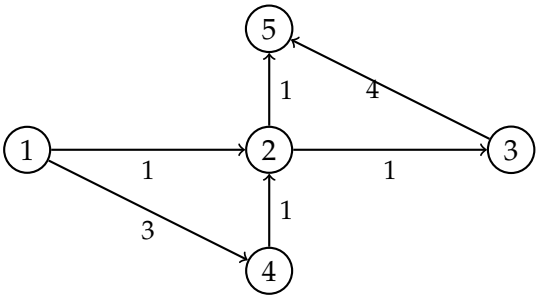


Figure 3.2.: European highway network (part) with higher resolution for the area around Germany (source: Own figure, data from (ITP/BVU, 2005))

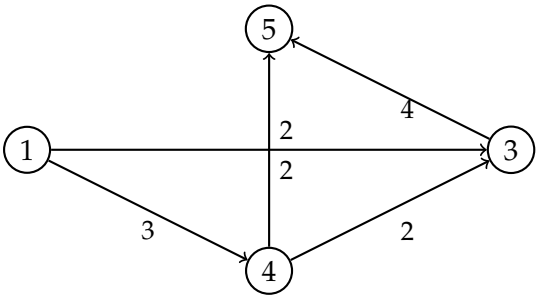
models a connection between two stations. In Fig. 3.2 parts of the European highway network are shown. Within and close to the boarders of Germany other road types, e.g., track roads, are included, thus the area is nearly black. The network is modeled as a directed graph. Each edge represents a road segment. A vertex represents a decision point where a traveler has to decide which road segment to travel next. So far, our notion of graphs state that vertices are only identified by their label. For the representation of the highway network, a notion of space is required. For each vertex an attribute can be attached specifying its geospatial location. Such attributes can be added without any modification of the graph structure similar to the example of the weight function for edges. Thus, vertices get a geospatial interpretation and can be located in space. The course of the road segments, however, is not specified. Vertices have coordinates, Edges not. Thus, the course of road segments between vertices may be inaccurate. If, e.g., speed limits or the number of lanes are attached as attributes to edges, a vertex can also represent a point in space where one of these attributes changes.

Graphs & Transport Networks

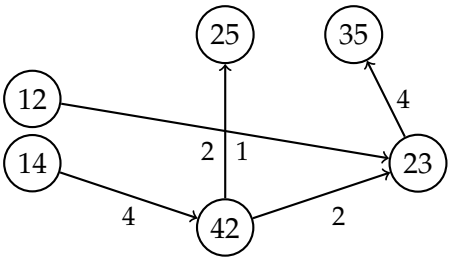
When large-scale, real-world problems shall be analyzed on top of graphs some problems emerge. Consider a change of the resolution for the highway network of France in Fig. 3.2. The network shall be modeled on the same level of detail that is used for



(a) Transport network as graph



(b) Directed graph for the transport network in Fig. 3.3a if left turns are not allowed on the edge from node 1 to 2



(c) Line graph for the transport network in Fig. 3.3a if left turns are not allowed on the edge from node 1 to 2

Figure 3.3.: Transport networks and graphs

the area around Germany. New vertices and edges are added to the graph. Others might be removed as they are replaced by a more detailed representation. Thus, the resulting graph is not comparable to the graph on the lower resolution. Additional attributes, e.g., specifying traffic counts, facility locations, or activities, might be attached to the vertices and edges of the original graph via attribute functions. These vertices and edges may no longer exist in the refined graph. The attributes are no longer well-defined and must be rematched. One may construct algorithms for approximation (Balmer et al., 2005a), but, on large-scale networks one may not be able to retrieve good parameter sets. The rematching is often done (semi-)manually with big effort. Thus, transport networks have a long life, a change of resolution might not be desired[1].

Graphs & Attributes & Semantics

Attributes are frequently attached to vertices and edges of a graph[2]. In our context, their application requires some careful interpretation. Consider the graph in Fig. 3.3a as representation of a transport network. Vertices are labeled, while the attributes on

[1] Data in semantically well-defined formats suited for machine processing could solve some of these problems. E.g., see `www.opengeospatial.org/standards`, last access 17.11.2013, for potential approaches.

[2] e.g. `gexf.net`, last access 14.10.2013

the edges specify the travel time. Let vertex 2 represent a junction. The shortest path between vertices 1 and 5 leads along vertex 2 and could, e.g., be calculated by the shortest path algorithm of Dijkstra (1959). Then, by some reason, left turns are forbidden from edge $\{1,2\}$ to edge $\{2,5\}$. An attribute function can be added to the edge that specifies the restricted left turn. Dijkstra's algorithm is not specified for left turn restrictions. One could modify the shortest path algorithm to take turn restrictions into account. Then, however, one has to proof correctness and care about efficiency. If no formal proof of correctness is provided, the modification of Dijkstra may exhibit the problems illustrated in Fig. 3.3. It is then better described as a "some path creation heuristic".

Another solution can be applied. The shortest path algorithm needs no modification even if turning moves are specified for a transport network. The turn restrictions must be reflected structurally, not by attributes. Fig. 3.3b shows a graph for the transport network in Fig. 3.3a that reflects the turn restriction. Dijkstra provides a correct shortest path, but vertex 2 is removed. Alternatively, a *line* graph can be used, if during the derivation of the line graph the turning move constraint is considered. The line graph considering the turn restriction is shown in Fig. 3.3c. Then, it is no longer obvious at which vertex of the graph the shortest path algorithm is started. In the abstract example presented here, one needs to run the algorithm twice.

Findings

The problem with attributes is not restricted to turn restrictions. It also occurs if certain edges are restricted to specific modes or vehicle types, e.g., bus or bicycle lanes. Concluding, use of graphs as representation for transport networks requires a careful selection of attributes. A shortest path algorithm must explicitly support these attributes. Alternatively, the shortest path calculation can be set up on top of a line graph. If the semantics of attributes is captured for the creation of the line graph, standard shortest path algorithms can be applied.

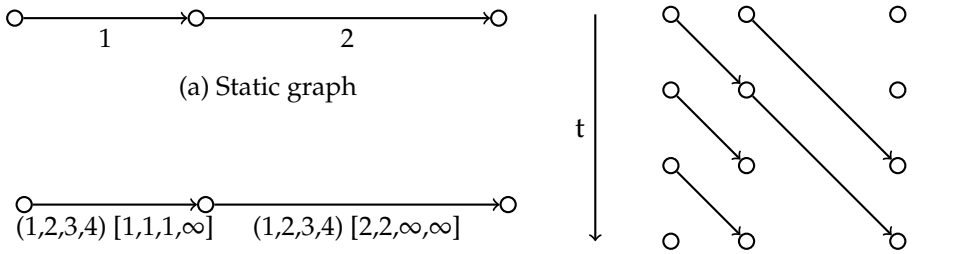
3.1.1. Transport Networks & Time

Dynamic Networks

Problems on transport networks often possess a dynamic nature. Modeling may require a discrete notion of time. In terms of network modeling this is challenging as over time streets may be closed or (re-)opened, transit connections may (dis-)appear, speed limits may vary, etc.

Time Expanded Graph

Dynamic transport networks can be modeled as *time expanded graph* (Ford and Fulkerson, 1962; Pallottino and Scutella, 1998; Köhler et al., 2009). A time expanded graph replicates the vertices of the static graph for each discrete time step. Each edge of the time expanded graph then connects two replicated vertices. These vertices are selected according to the dynamic attribute function, e.g., the required travel time to traverse an edge. Fig. 3.4a shows a static graph with travel time attributes on edges. The corresponding time expanded graph is shown in Fig. 3.4c. A time expanded graph is still a graph in the sense of graph theory, that is, solution algorithms, concepts, and proofs still apply. Thus, time expanded graphs permits an elegant modeling of dynamic transport



(a) Static graph

(b) Time aggregated graph

(c) Time expanded graph

Figure 3.4.: Dynamic graph representations for $T \in [1, 4]$

networks, provide a general problem formulation and many efficiency considerations or optimizations from graph theory can be applied with little effort. If the problem is periodically the time expanded graph can be modeled cyclically (e.g. Köhler and Strehler, 2010).

Graph Size, Data & Memory

Obviously, the size of time expanded graphs increases in the length of the time interval and the discretization of time. In an asymptotic notation (e.g. Russel and Norvig, 2010) the time expanded adjacency list representation of a static graph $G = (V, E)$ requires memory of order $O((|V| + |E|) \cdot T)$, where T is the number of discrete time steps. Thus, in terms of data modeling and memory consumption time expanded graphs are not a good choice for large-scale applications. If each vertex or edge of G possesses many attributes that vary over time independently from each other the problem can no longer be modeled as time expanded graph. George and Shekhar (2008) address this problem in detail. A *time aggregated graph* representation is proposed. Attributes of vertices and edges can vary over time. Furthermore, vertices and edges may dis- or (re-)appear over time. Time variance is modeled via time series that are attached to edges and vertices. Each series contains the values of a time variant attributes. Fig. 3.4b shows an example of a time aggregated graph. The semantics is equivalent to the time expanded graph in Fig. 3.4c. The time series do not necessarily cover all discrete time steps, their length is restricted to the validity of the attribute. Let α be the average length of all time series. Then, the proposed representation requires $O((|V| + |E|) \cdot \alpha)$ memory. Thus, as long as $\alpha < T$ the representation outperforms the time expanded graph. Following George and Shekhar (2008), standard shortest path algorithms can be applied on their representation.

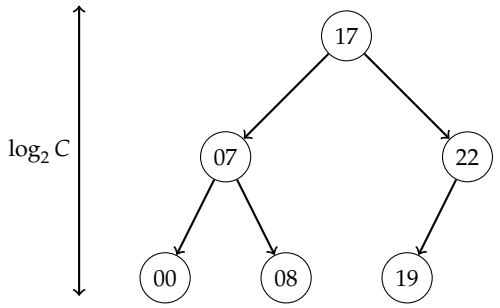
3.1.2. Time Dependent Attributes

Simulation

In principle, MATSim uses a static, directed graph for the representation of the transport network. Edges depict road segments while vertices can be interpreted as decision points in space that have a coordinate as attribute. The location of vertices in space is not supposed to vary over time. Edges represent space, all other attributes relevant for the domain are attached to edges.

Time [hh]	Speed Limit [km/h]
00	∞
07	150
08	130
17	110
19	150
22	∞

(a) Time dependent attribute of an edge



(b) Binary search tree for representation of a time variant attribute with C changes

Figure 3.5.: Time dependent network

Evacuation

For MATSim, a model for time dependent attributes exists. The model is derived from an use-case in the area of evacuation simulation. Large-scale inundations or conflagrations do evolve over time and do not cover all road segments of a transport network at once. Once they are affected by the disaster they cannot be used for evacuation purposes. For details of the evacuation context and implementation hints readers are referred to (Lämmel et al., 2010). The following reviews the conceptual backgrounds of the model.

Concepts

In the simulation context, inundation on road segments can be modeled via time variant attributes for edges. Either the maximum outflow, or the speed-limit attributes of an edge can be set to 0. It is not necessary to remove vertices and edges from the graph. As all action is modeled on edges, vertices do not need further attributes. Instead of time expanding the graph or storing attributes as time series, the changes of time varying attributes are recorded. Each change has an associated time stamp and may be relative to the previous value or absolute. Fig. 3.5a illustrates a speed limit of an edge that varies over time. The resulting time variant attribute value can then be stored in a self-balancing binary search tree. A binary search tree is a data structure that can be displayed as another special type of graph. Fig. 3.5b shows an example for the time varying speed limit in Fig. 3.5a. The organization of the data structure permits efficient access to its vertices. In the example of Fig. 3.5b all labels left of vertex 17, the root, are smaller than 17. The very reverse holds on the right side of the root. In our example, this order is applied recursively, top to bottom. Thus, access times to vertices are bound to the depth of the binary search tree. A binary search tree is called self-balancing if the data structure ensures that the depth is always minimal. Thus, stored in a self-balancing binary search tree access time to a time dependent attribute with C changes is bounded by $O(\log C)$.

Space & Time Complexity

Memory consumption of binary search trees is linear in the number of entries, i.e., $O(C)$. Thus, overall memory consumption of a static graph with changes on edges is $O(|V| + (|E| \cdot C))$. In case of a large number of time steps T and attributes that stay constant over long time series and seldom change their value, i.e., $C < T$, $\alpha \approx T$, this time dependent

network model requires less memory than the approaches of the previous section. Access to attributes is rather expensive ($O(\log C)$). If the number of time variant changes is constant at runtime, access to attributes can be optimized by use of arrays and binary search (Lämmel et al., 2010). Otherwise, reorganization costs for the binary search tree must be considered.

3.1.3. Discussion & Findings

Life Cycle

Transport networks and their underlying graph may have a long life cycle. As the variety of data to create such graphs is wide, there is no standardized method to derive a graph from data automatically. After derivation of a graph, manual modifications are frequently required to ensure a meaningful output of a simulation model. Recreation of the graph may not be desired, as the attributes matched to the network have to be rematched.

Attributes

If standard algorithms from graph theory are applied on top of the transport network, attributes have to be added carefully. If the semantics of attributes is considered the transport network may no longer be a graph. For shortest path calculations this may be resolved if a line graph is used for shortest path algorithms. The line graph has to reflect the semantics of attributes.

Evacuation

For simulation in an evacuation context, the time dependent attribute representation is well suited. When an edge is once covered by an inundation it is not supposed to change further. The number of changes is rather limited. A successful use-case is presented in (Lämmel et al., 2010).

Traffic Signal Control

Traffic signals impose time variant attributes to a transport network. The approaches presented in the first sections of this chapter may be used for modeling. If traffic signals are controlled by a fixed-time control, the problem is periodical. A cyclical time expansion of the network can be applied (Köhler and Strehler, 2010). However, for large-scale applications memory consumption of time expansion and the resulting network size still limits analysis to subnetworks, see Sec. 5.4.

In case of a traffic-responsive signal control, a periodic formulation is no longer suitable. The approach by George and Shekhar (2008) requires too much memory. Instead, the time dependent attributes developed for evacuation scenarios might be considered. For traffic-responsive signal control, the number of changes is clearly higher than for evacuation scenarios. The number of changes, C, to the network should be small. Otherwise, lookup costs increase logarithmically and memory consumption increases linear in C. In case of a traffic-responsive signal control a preprocessing of changes is not feasible. Then, the time dependent attributes have additional, permanent reorganization costs for the binary search tree data structure. Thus, in the following, potential extensions of the traffic flow model of the mobility simulation are considered.

3.1.4. Nomenclature

Nomenclature

To reflect the subtle differences between transport networks and graphs, two different nomenclatures are applied: The terms *edge* and *vertex* refer to a well-defined, directed graph. If the notion of a well-defined graph is not important, edges are referred as *links* while vertices are called *nodes*.

3.2. Queue Models for Traffic Flow

Traffic Flow Models

Popular models for the simulation of traffic flow on roads are, e.g., "car following models" (Wiedemann, 1974; Gipps, 1981) or the "Nagel-Schreckenberg model" (Nagel and Schreckenberg, 1992). These models share a common characteristic: they are computationally relatively expensive. Therefore, in the domain of transport engineering, queue models have been developed (Gawron, 1998b; Simon et al., 1999; Cetin et al., 2003; Cetin, 2005; Cremer and Landenfeld, 1998; Charypar, 2008).

Queue Models

Queue models disregard most of the details of vehicle movements on a road segment. The traffic network is modeled as directed graph. The interpretation of links differs from model to model and refers to some part of a road.

3.2.1. "Fast Lane" Model by Gawron

"Fast Lane" is a queue model that is explicitly designed for high speed mesoscopic traffic simulation (Gawron, 1998a,b). Vehicles entering a link have to stay on that link at least as long as they would travel at their desired velocity v_0. During this time no computation needs to be done, the vehicles are stored in a priority queue. Afterwards the vehicle is placed into one of several point queues. A point queue can contain an infinite number of vehicles and has no space restriction. One point queue is used for each downstream link. These point queues jointly restrict the outflow of the link; the documentation remains unclear as to how the joint link capacity is shared between the point queues. The joint link capacity is modeled by a normal distribution that is cut off at negative values. The number of vehicles that can be on a link simultaneously is restricted to a maximum of $c_{storage} \in \mathbb{N}$. A vehicle is moved to the downstream link if there is space available, i.e., the number of vehicles is less then $c_{storage}$. This enables the modeling of spill-back. Gawron states that the links can, in principle, model signalized intersections. However, there are no further specifications.

3.2.2. Extension of "Fast Lane"

The traffic flow simulation of MATSim is based on Gawron's model. However, the model was modified at some points:

- Intersection Logic: In "Fast Lane", approaches of an intersection are processed in a fixed sequence. Thus, if spill-back occurs, some links are served with a higher priority than others. Therefore, the model was extended by a probabilistic prioritization of approaches to an intersection (Cetin et al., 2003; Cetin, 2005).
- Turning Moves: Instead of using a point queue for each turning move, the MATSim model uses only one queue for all turning moves (Cetin, 2005).
- Speed limit, v_{fs}: To calculate the minimum time a vehicle has to stay on a link tt_{fs}, the MATSim model uses a speed limit attribute for each link instead the desired velocity of vehicles (Cetin, 2005): $tt_{fs} = l/v_{fs}$ This change was undertaken due to some artifacts of the model in case of spill-back: In "Fast Lane" vehicles with a high desired velocity can overtake vehicles that have a low desired velocity even in situations where the link is completely jammed.
- Flow capacity, c_{flow}: The random draw from a normal distribution in "Fast lane" and the random draw proposed in Simon et al. (1999) to model flow capacity is replaced by a deterministic version. As vehicles cannot be divided into parts (number of vehicles $\in \mathbb{N}$), one has to consider the case where $c_{flow} \notin \mathbb{N}$, i.e.,

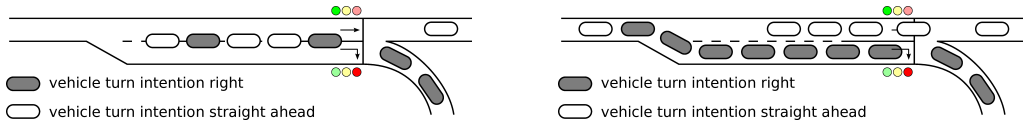
  $$c_{flow} = floor(c_{flow}) + frac(c_{flow}),$$

  whereby $floor(x) := \lfloor x \rfloor$, $frac(x) := x - \lfloor x \rfloor$, $x \in \mathbb{R}^+$. In each simulated timestep, $floor(c_{flow})$ vehicles may leave the link. Fractional capacity $frac(c_{flow})$ is accumulated per time step until this is sufficient for an additional full vehicle that may then leave the link. This change was done to improve the simulation of small samples of demand: For prototyping or sketch planning usually 1 % or 10 % samples of the transport demand are used in order to save computation time. A 1% sample, together with a flow capacity of, say, $900\,veh/h = 0.25\,veh/sec$, leads to a flow capacity for simulation of $0.01 \times 0.25\,veh/sec = 0.0025\,veh/sec$. Random draws based on such a small probability leads to very large fluctuations. Link travel times get very unpredictable for vehicles.
- Storage capacity, $c_{storage}$: According to Simon et al. (1999), the maximal number of vehicles on a link is retrieved by

  $$c_{storage} = l \cdot n_{lanes}/l_{site},$$

  whereby n_{lanes} is the number of lanes represented by the link, l its length, and l_{site} the inverse of the jam density, $l_{site} := 7.5$ meters.
- Calibration: To calibrate the model to small sample sizes, two scaling parameters are available: α_{flow} and $\alpha_{storage}$. The effective flow capacity of each link in the network can be scaled to $\alpha_{flow} \cdot c_{flow}$, the storage capacity to $\alpha_{storage} \cdot c_{storage}$.

These changes have been subject to projects prior to this work. As there is not enough evidence, they are not changed.



(a) Single queue, spill-back is not captured correctly

(b) Multiple queues, spill-back is captured correctly

Figure 3.6.: Influence of traffic signals on traffic flow and spill-back can be modeled by a queue model, if the layout of turn pockets is considered.

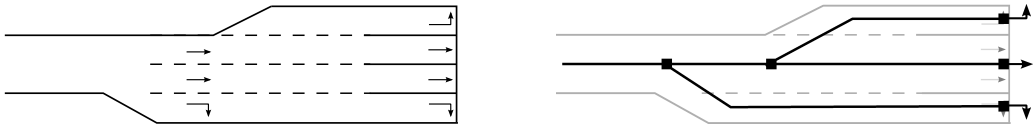
3.2.3. Mesoscopic Traffic Signal Simulation by Cremer and Landenfeld

Cremer and Landenfeld (1998) propose a mesoscopic traffic model focusing on the modeling of signalized intersections. The principles of the model for the movement on a single link are quite similar to "Fast lane". A link is defined as connection between two junctions. Relevant differences are a simple specification for vehicle movements on all parts of the link, and a logic to capture unprotected left-turns. Furthermore, the maximum flow of links is set up via the sampling time of the model, i.e., simulated flow rates at signalized intersections have plausible values only if a uniform velocity of 50 km/h and a sampling time of 2 seconds are used. The calibration of the model via the sampling time implies that flow rates are equal for all links. Waiting queues for distinct turning movements, including their spatial extension are modeled explicitly. For that reason, in case of spill-back mutual blocking effects between several turning directions are captured. This is important if traffic signals are simulated microscopically, see Fig. 3.6: If a single queue is used (Fig. 3.6a), the first vehicle blocks all other vehicles upstream. This can capture reality if the approach has only one lane for all turning moves but does not hold in all cases. In the case, however, that the approach has several lanes for signalized turning-moves, a single queue model distorts the effects of signalization. In contrast, Fig. 3.6b shows the modeling approach from Cremer and Landenfeld (1998). Vehicles with distinct turn intentions do not block each other until the available space for queueing on the lane is used completely.

3.2.4. Modeling Traffic Signals

Traffic Flow Model

The traffic flow models reviewed in the last sections are designed to simulate network wide traffic efficiently. The Cremer and Landenfeld model has a well-defined semantic how turn pockets can be represented to capture effects of spill-back at signalized intersections. But, the model has some drawbacks. First, the modeling of vehicle behavior on all parts of the link needs additional computation time. Gains in expressiveness of this part of the model are limited, as the main use cases are visualization and to provide meaningful sensor information. This information can still be calculated if needed. Second, the flow restrictions of the links are determined via the sampling time of the



(a) Typical real road layout

(b) Part of the graph required to model the road layout

Figure 3.7.: Transition from a real road segment to a graph layout

simulator and are homogeneous for all links. The assumption of homogeneous flow at all intersections makes it difficult or even impossible to calibrate large-scale scenarios. In addition, most traffic signal control strategies update each second and not every 2 seconds.

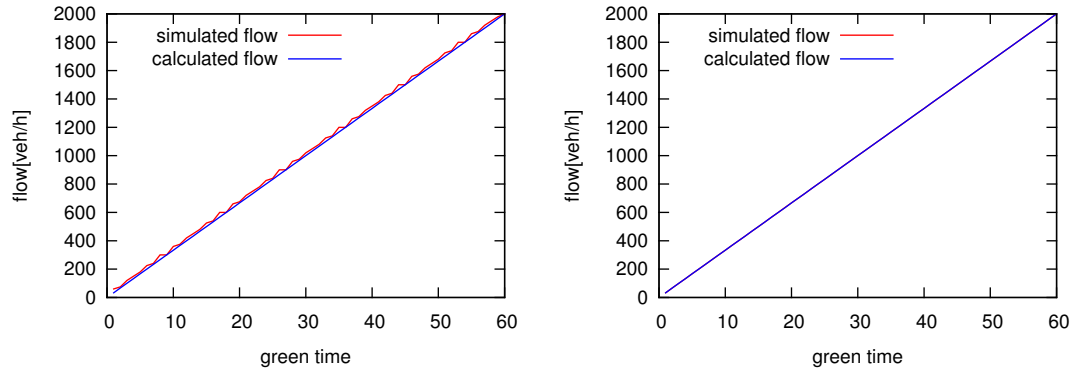
Thus, here the traffic flow dynamics is taken from the extended Gawron model. If the graph layout that defines the transport network can be changed, turning lanes can be represented adequately (Gawron, 1998b, p. 37). Fig. 3.7a illustrates a typical layout of a real-world road segment with several turning lanes at its end. The layout of the corresponding graph is shown in Fig. 3.7b. If each edge is represented by a link of the Gawron model spill-back effects between turning lanes are captured.

Traffic Signals

In the Highway Capacity Manual (Council, 2000, p. 16-14) the capacity of a signalized lane C_i is defined as $C_i := f_i \cdot q_{Si}$, where f_i is the percentage of green given to lane i and q_{Si} is the saturation flow, if traffic signals are switched off multiplied by several correction factors.

The extended Gawron model can capture flows at signalized intersections by modifying the maximum permitted outflow c_{flow} according to green time of traffic signals. If the signal shows a color that allows vehicles to leave the link, flow is permitted with $floor(c_{flow})$ while $frac(c_{flow})$ is accumulated. If driving is not permitted, flow is stopped. Also, accumulation of the fractional part of c_{flow} is stopped. If driving is permitted for n timesteps, a maximum flow of $n \cdot floor(c_{flow}) + n \cdot frac(c_{flow}) = n \cdot c_{flow}$ is allowed. Thus, c_{flow} is a calibration parameter that has a similar interpretation as q_{Si} in the Highway Capacity Manual.

For the accuracy of the model it is important to stop accumulation of $frac(c_{flow})$ when driving is not permitted. This is illustrated in Fig. 3.8 that shows simulation results for a typical signalized link within an urban area, i.e, $c_{flow} = 2000\,veh/h$. For one hour, each second one vehicle enters the link. The green time is varied from $1\,sec$ up to the cycle time of $90\,sec$ and displayed on the x-axis of the figure. The y-axis shows the simulated number of vehicles leaving the link. The flow calculated by $C_i = f_i \cdot q_{Si}$ is depicted by the blue curve. The red curve shows results of the simulation if flow accumulation is not stopped during red time. Compared to the calculated values, there is too much simulated flow. Furthermore, the curve shows some plateaus that are not specified by



(a) All time accumulation of $frac(c_{flow})$

(b) Accumulation of $frac(c_{flow})$ when flow is permitted

Figure 3.8.: Theoretic calculation vs. simulation results

the model. Fig. 3.8b shows the same situation for a model that stops accumulation of $frac(c_{flow})$ during red time. Differences are not observable.

3.2.5. Lanes

Lanes

So far, traffic signals can be modeled straightforward by Gawron's "Fast Lane" or the extended version implemented in MATSim. The situation becomes more complicated, if the graph structure of the transport network cannot be changed. To resolve this problem, the implementation in MATSim allows the modeling of a subgraph on top of a link that reflects the structure shown in Fig. 3.7b. Edges of the subgraph are called *lanes*. Traffic flow on each lane is simulated nearly the same way as for links in the extended queue model. Just the calculation of minimum travel time slightly differs to avoid systematic errors of temporal resolution (Gawron, 1998b, p. 38). The calculation for lanes is set up in a way that is compatible to a link of the extended queue model without lanes, i.e., in freeflow conditions both models result in exactly the same travel times. This avoids further propagation of systematic errors. At the beginning of the link only one lane may exist. The next lane is determined by the necessity to be in the correct turning lane for the next downstream link of the vehicle's precomputed route. According to the Cremer and Landenfeld approach, the vehicle is placed on the lane that currently contains the smallest number of other vehicles if there are several lanes leading to the same downstream link. Note, that the lanes of the model have no 1:1 relation to the lanes existing on a link in reality. The use of lanes implies a specification of the downstream links, thus, specific turning moves can be forbidden. This requires a modified routing of vehicles.

3.2.6. Routing

Lanes & Routing

Within the overall simulation process introduced in Chapter 2 the use of lanes affects the routing module. Routes are specified within MATSim as sequence of links[3]. The shortest path algorithm is set up on link travel times. These do not reflect the travel time differences for turning moves modeled by the subgraph. In addition, turn restrictions cannot be captured by standard shortest path algorithms. To avoid these problems, the shortest path algorithm can be set up on the line graph. Thereby, the turn restrictions are considered when the line graph is created. The dynamic link travel times for the line graph reflect the link travel time on the original network plus the travel time for the specific turning move. Then, the shortest path calculation captures the effects of lanes without further modification.

3.2.7. Discussion

Physical Queues

Capturing the physical dimension of spill-back is a key feature of queue models. A badly designed signal control strategy can quickly lead to grid lock in the network due to the resulting spill-back (Sec. 5.1). Thus, modeling of spill-back is considered important, if traffic signals are simulated microscopically. For certain road layouts a microscopic representation of turning pockets is needed. With the model proposed by Gawron, this can be modeled straightforwardly. The same applies to the extended version implemented within MATSim. Thus, the number of required concepts is kept small. The network modeling process, however, can get more complicated.

Network Modeling

Many different attributes may be attached to the links of a transport network, e.g., traffic count data, transit stops, transit lines, speed limits, etc. Geospatial location may not be sufficient to describe the matching between attributes and links. Often, certain attributes are matched manually. If, for the representation of turn pockets, the network layout is changed, the manual matching must be repeated. This can result in huge effort. Further, a comparison between a simulation with and without an implicit model for turn pockets is difficult. Simulation models deliver different results for different network resolutions (Gawron, 1998b, p. 38). For comparison, one needs some algorithmic that traces the changes in network structure. If the network is changed to capture turn pockets, the shortest path algorithm is responsible to select the appropriate turn pocket on a route. If many turn pockets lead to the same downstream link, the number of required iterations is increased.

Lanes as Concept

These drawbacks can be resolved in part, if lanes are modeled as separate concept. A lane layout can be attached to a link as optional attribute. Attributes matched to the link are still valid. The routes defined on the network still refer to links. Shortest paths are calculated on the line graph and reflect different travel times for single lanes and turning moves.

[3] In older version of MATSim, this can also be specified by a sequence of nodes.

For MATSim, both versions are implemented. One can change the network layout and attach traffic signals to links or use the lanes as concept on top of links.

3.3. Findings

The chapter illustrates, why attributes attached to links of transport networks should be treated carefully and in conjunction with their semantic interpretation. Then, several options for the modeling of traffic signals are discussed. The most promising solution is an extension of the traffic flow model. In principle, Gawron (1998b) can be applied, but, the documentation for traffic signals is not very precise. Further, the implemented version is not equivalent to Gawron (1998b) due to a sequence of subsequent extensions (Simon et al., 1999; Cetin et al., 2003; Cetin, 2005). The chapter reviews and summarizes the extensions and provides a detailed documentation how traffic signal control can be represented by queue models for traffic flow.

Applications and studies using the queue model for traffic signal control are presented in Chapter 5. Moreover, in Chapter 6, the queue model approach is applied to the simulation of air transport technology.

Chapter 4

Software Engineering & Design

4.1. Introduction

"Designing object-oriented software is hard, and designing reusable object-oriented software is even harder. You must find pertinent objects, factor them into classes at the right granularity, define class interfaces and inheritance hierarchies, and establish key relationships among them. Your design should be specific to the problem at hand but also general enough to address future problems and requirements. You also want to avoid redesign, or at least minimize it..."

Gang of four (Gamma et al., 1995)

Methodology for object-oriented (OO) software design is available and well documented in many sources (e.g. Gamma et al., 1995; Fowler, 2004b; Martin, 2009). In the past decades, several concepts and considerations were established for software design — *Modularity*, *Information Hiding*, *Abstraction*, *Reusability*, *Usability* or *Extensibility* are prominent examples. These goals are partially competitive. Following solely the concept *Abstraction* might lead to the development of a *General Problem Solver*, an approach from the early ages of artificial intelligence that contributed to AI-planning (Russel and Norvig, 2010, p.393) but was also criticized for its misleading name (McDermott, 1976). Even the question of the general problem is considered irresolvable. Thus, one ends in a "specific question" for a "specific general problem" that can be solved by a single software. In respect to software design, this implies some trade-offs and design decisions that are unique for each piece of software. In the following, trade-offs and methodologies of software design for MATSim are discussed.

MATSim = ?

Multi Agent Transport Simulation implies that a number of concepts are used. For each of them, agent-based (transport) models, transport simulation, and forecasting models many books and publications exist (e.g. Weiss, 1999; Russel and Norvig, 2010; Nagel, 2005). In this work, MATSim is considered as a software that enables usage of default functionality to answer transport analysis and forecasting problems on a high, agent-based resolution, but for large-scale applications. Large-scale implies some level

of abstraction to enable computationally affordable simulation and problem solving. Multi-agent based transport simulations are often implemented by separating a simulation of physical constraints from a simulation of agents. Agents interact with the physical constraints, e.g., as travelers in the transport system, or as control algorithms for traffic light control. Interaction may change certain aspects of the physical constraints. Yet, certain constraints should not be violated; for example, synthetic persons should be required to consume time while physically moving from one location to another rather than being teleported in zero time (which is possible in the computer but not in reality).

Community

Developed by different groups of developers, MATSim is the work of many; at the time of writing main developers are located at ETH Zurich, senozon AG, and TU Berlin. Even more heterogeneous is the group of users. In consequence there are several, different, domain-specific perspectives on the software.

Users

Users of MATSim typically come from transport and civil engineering domains and are aiming to solve problems for which the agent-based modeling approach is more suited than more traditional four step process modeling. The typical use-case starts with the collection and preparation of data. Then, several simulations are run on a cluster for several days and compared afterwards. Often, this must be repeated several times as data may contain errors that do not become visible until simulation results are retrieved. If no automatic calibration is available, repetition of simulation runs might be needed to calibrate the model. Multi-agent simulation pays off when heterogeneous user preferences, time dependent user reactions, or/and microscopic modeling of public transport are studied, (e.g. Rieser et al., 2008; Grether et al., 2009b; Neumann and Nagel, 2010). At the time these studies were undertaken, there was no default software support for the required functionality; MATSim had to be extended by custom model components. Today, some of this functionality is available in MATSim or as a contributing project. However, modeling problems at this high level of detail make it hard to define a default methodology and implementation that suits every user. So, a user quickly finds himself in the second group, the researchers.

Researchers

Advanced users may reach a point at which the model implemented in MATSim is not sufficient to answer their questions. Then, they start researching extended or different modeling approaches. In many cases, these models shall be applied to or implemented with MATSim. Currently, such work is done, e.g., in the field of evacuation (Lämmel and Flötteröd, 2009; Flötteröd and Lämmel, 2010), destination choice of travelers (Horni et al., 2012) the planning and analysis of urban development in cooperation with the UrbanSim project (Nicolai et al., 2011), or models enabling the simulation of electric vehicles (Waraich et al., 2009b). Addition of functionality to the basic algorithm or changes to small parts of it are sufficient in most cases. A strong computer science background is helpful but cannot be expected. But researchers must be able to implement their model. How far this software based modeling is trustworthy in terms of good scientific practice is still subject to ongoing discussions (Joppa et al., 2013). Making modeling frameworks like MATSim more modular, reusable, and expandable can reduce the lines of code to

be rewritten for a different modeling approach. Thus, the confidence in the scientific contribution may be improved.

Computer Science & Engineering

From a computer science perspective, MATSim can be seen as substrate delivering plausible, network wide traffic patterns, that can be used to test agent-based concepts (e.g. Bazzan et al., 2008; Bazzan, 2009). Furthermore computer science may provide programming languages and methodology that make extension of software as simple as possible. Choosing the most suitable approach, set-up of methodology, and implementation of core concepts is the task of software engineering. Optimally, this should be invisible to other users.

Usability & Programming

Unfortunately, invisibility is not reached. This chapter explains parts of the methodology and set-up that was chosen in the last years in order to improve the first prototypical 2007 version of MATSim. Motivated by the different perspectives on the software the main goals are *Modularity*, *Reusability* and *Extensibility*. As most users come with a non computer science background practical solutions are required. Full knowledge of Gamma et al. (1995); Fowler (2004b); Martin (2009) cannot be expected. If the software architecture is based on a subset of these concepts, the overhead to understand the underlying software engineering is reduced. The following reviews and explains the chosen subset. Two software design patterns, i.e., *Abstract Factory* and *Observer* (Gamma et al., 1995), in conjunction with some other software design approaches improve the main design goals. As an example for an extension of the MATSim core algorithm, the chapter shows how a traffic signal implementation can be attached to the simulation.

Overview

The rest of the chapter is organized as follows: First, in Sec. 4.2 the main thoughts and used concepts from Gamma et al. (1995) are reviewed. Readers aware of the book can skip most of the section. In addition, the section reviews other methods for software design. Sec. 4.3 sketches the central algorithm and outlines the state of MATSim when this work was started in 2007. Then, Sec. 4.4 and Sec. 4.5 discuss some key issues of extensibility for OO software and explain consequences by example. In Sec. 4.6 the design of MATSim at time of writing is explained, while Sec. 4.7 shows how an extension for the simulation of traffic signals can be designed by the same principles and used as plugin for MATSim. The chapter ends with a discussion and conclusion.

Please note that Sec. 4.6 reuses material from Grether and Nagel (2013b).

4.2. Software Design & Development

In his code refactoring handbook, Fowler (2004b, p. xiii) states *"Design patterns provide targets for refactoring"*. This section shortly reviews terminology and relevant design patterns from Gamma et al. (1995) to provide a basic foundation for the subsequent sections. Afterwards some other principles of OO software design are collected and reviewed.

4.2.1. Object-Oriented Software Design

Design Pattern

Solutions to frequent software design problems for OO-software are collected in Gamma et al. (1995). Each solution is a *design pattern* that describes an abstract template for a software structure. Concrete implementation may differ in detail with respect to the programming language. The design patterns can be used independently from the chosen language.

Design patterns have a unique name and are categorized by scope and three different pattern-purposes: 1) creational, 2) structural and 3) behavioral. Categorization helps to understand and select specific patterns. The names of the patterns provide a common namespace for tools. For each pattern motivation, documentation and solutions for example problems are available. Thus, Gamma et al. (1995) provide not only solutions to common problems, they define a namespace for the tools in the box. The concrete solution to a software design problem can be seen as selection, combination and implementation of certain tools. Selection and combination of design patterns need some care. Use of too many patterns may complicate design, increase maintenance costs, and result in a loss of performance.

Before the design patterns of interest are reviewed, some terminology and principles require a short introduction.

Terminology

To solve OO design problems independently from the programming language an abstract terminology is required. The following shortly introduces the nomenclature from Gamma et al. (1995).

Interfaces

The basic idea of OO programming is the specification of classes of objects. Objects support certain operations. An operation's *signature* consists of its name, parameters and return type[1]. An object's *interface* is the set of all signatures provided by the object. The interface of an object is the only way to communicate with the object.

Types & Inheritance

A *type* is a name for a specific interface. Types can consist of a subset of the signatures provided by an object. Thus, one object may have many types. Types can be organized hierarchically: A *subtype* contains all operations of its parent type. If a type is derived from a *supertype*, this is called *type-inheritance*. Objects of a subtype can be used at all places where the supertype is required because they fulfill, by construction, the interface of the supertype.

Polymorphism

A request sent to an object must match one operation of the object's signature. The implementation of the operation may differ between objects while the associated types stay fixed. Which implementation is executed is determined at run-time, this is called

[1] This is the abstract definition of signature by Gamma et al. (1995). Interpretation in Java is different

dynamic binding. Thus, objects that match the same type may be substituted, this is called *polymorphism*.

Classes & Inheritance

Each object is an instance of a certain *class* and can be created by instantiating the class. Classes specify the set of operations, their implementation, and the internal data of the object. Classes can be organized hierarchically in analogy to types. A class may be subclass of a parent class, i.e., all data and operations of the parent class are available in the subclass. This is called *class inheritance*.

Classes can be *concrete* or *abstract*. One cannot instantiate objects from an abstract class. Abstract classes define a part of their subclasses' interface and may provide implementation of operations. All other classes are called concrete.

Classes and Types

The operations of an object must be implemented in the class defining the object. Thus, the class definition implicitly defines one type of the object. Objects, however, may have many other types. If two objects are of same type they do not have to be created from the same class.

Class vs Interface Inheritance

Some design patterns depend on the *distinction between class and interface inheritance*. Class inheritance means inheritance of implementation. Its main purpose is to avoid reimplementation of already existing operations. In contrast, interface or type inheritance defines relations between types. By specifying inheritance between two types nothing is said about the implementation. Many programming languages, however, do not make a clear distinction between type and class inheritance. While expert programmers are aware of the distinction, for non-experts this is often confusing. The interplay between the different types and their type inheritance hierarchy is essential for many design patterns. Some patterns may not work if the type of an object is only defined by the implementing class.

Principles of OO Software Design

First Principle – Use Interfaces

Gamma et al. (1995) propose two principles of OO software design. The first is *"Program to an interface, not an implementation"*. Main motivation for the principle is to enable polymorphism and to clearly separate a client's needs for certain operations from potential implementations. This leads to reduced dependencies between different subsystems of software and thus reduces maintenance effort. Creational patterns can help to enforce this principle

Second Principle – Use Object Composition

The second principle is *"favor object composition over class inheritance"*. Object composition denotes the creation of new functionality by assembling already existing objects. To make code reusable, class inheritance can be used. Class inheritance is often called white-box reuse as the inheriting class' internals are often visible to the subclasses and thus breaking encapsulation. Changes in the superclass mostly imply changes in subclasses. In contrast, object composition needs no information about the internals of the

```
public class B implements A {
  private A delegate;

  public Object operationFromA(){
    return delegate.operationFromA();
  }
}
```

Listing 4.1: Sketch of a delegation implementation in Java

objects to assemble and is called black-box reuse. Class inheritance is defined at compile time, object composition, however, at run time. Thus, object composition is more flexible and ensures encapsulation in contrast to class inheritance that has the advantage to be more straightforward for many non-expert programmers.

Delegation

Delegation is a special way of object composition that is considered as powerful as class inheritance. Instead of subclassing a class A by class B, delegation means that B references an instance of A. Methods invoked on B that are already implemented in class A are bypassed to this instance of class A via the reference. Listing 4.1 shows the implementation of delegation in class B in Java assuming there is a appropriate interface for class A.

4.2.2. Design Patterns

As structural design patterns are not required in the subsequent sections, the following reviews the essential creational and behavioral patterns, only.

Creational Patterns

Essentially, the group of creational patterns consists of two patterns — *"Abstract Factory and Builder yield objects whose only responsibilities are creating other objects"* (Gamma et al., 1995, p. 13). The main advantage of creational patterns is the encapsulation of knowledge about implementation, thus encourage use of interfaces and types instead of concrete implementations ("products").

Builder vs. Factory

Creation of objects can be simple or complex. Simple means that only the concrete implementation for a type is specified. Complex build processes are used for the composition of an object by use of several other objects. For simple object creation *Abstract Factory* is the pattern of choice, for complex build processes Builder is used (Gamma et al., 1995, pp. 105–106).

Abstract Factory can be implemented with *Factory Methods*. The idea behind Factory Method is to avoid direct instantiations of objects (calls of the constructor) by placing

```
public class DefaultFooFactory implements FooFactory {

  public Foo createFoo(){
    return new DefaultFoo();
  }
}
```

Listing 4.2: Abstract Factory implementation in Java

this code in a separate method. Using Abstract Factory means to locate the Factory Method in a separate class. Listing 4.2 shows an example implementation of Abstract Factory in Java.

Behavioral Patterns

Following Gamma et al. (1995, p. 221), Behavioral patterns look at responsibilities and, even more important, at the communication between objects. Their focus is on object interconnectivity while control flow is hidden.

Object Communication

One of the most prominent patterns is the *Iterator* pattern that defines a standard to loop over elements of a container. Who had to think once for some seconds while balancing over array indices or pointers, may have made the experience that getting his head free for other coding issues clearly shows some advantages. Luckily, Iterator is nowadays standard concept of most programming languages and their syntax encourages use of Iterators.

Iterator

The *Observer* pattern, also known as *Model View Controller (MVC)* Pattern, is another example for a behavioral pattern. Most of the recently upcoming www application frameworks ships and advertises some kind of MVC implementation[2]. Observer intends to structure dependencies of object communication, i.e., by defining a *Subject* and *Observers*[3]. Modifications of the Subject trigger the notification of the Observers about the change. The Subject may have any number of Observers. Awareness of their concrete structure is not required. The Subject is just broadcasting change messages. Thus, the Observer pattern allows decoupling of classes because of the unidirectional communication between them – from the Subject to the Observers. The order in that Observers are notified is not specified by the pattern. Observers should be independent from each other. The undefined order of message broadcasting enforces the decoupling of Observers. If developers of the core Subjects are in a situation *"when a change to one object requires changing others, and you don't know how many objects need to be changed"* and *"an object should be able to notify other objects without making assumptions about who these*

Observer

[2] e.g., `http://framework.zend.com/`, last access 24.07.2013

[3] In the MVC nomenclature, the Subject is called Model while Observers are referred as Views.

objects are" (Gamma et al., 1995, p. 294). Use of the Observer pattern might be a good choice.

Visitor

The *Visitor* pattern intends to ease adding new operations to an OO type hierarchy while the classes defining the hierarchy should not be changed. The general idea is to define the new operation not on the type hierarchy itself but by the implementation of a parallel hierarchy of Visitors. After registering an Visitor instance at the original hierarchy, the new operation is invoked via a callback function to the parallel hierarchy. While *Visitor* eases to add new operations, its overall structure is rather complicated. Also, it complicates adding new elements to the original type hierarchy and may break encapsulation (Gamma et al., 1995, pp. 335).

Template Method

The behavioral pattern *Template Method* is applied in nearly each software that has more than one execution path. The general idea is to define an abstract algorithm whose concrete steps can be redefined in specific implementations.

4.2.3. The Java Programming Language

Gamma et al. (1995) provides example implementations of patterns in Smalltalk and C++. The concepts discussed in this chapter are implemented in the Java programming language. Readers interested in example pattern implementations for Java are referred to Metsker and Wake (2006). In the following, some important conceptual differences between Java and C++ are reviewed.

C++

In C++, classes and abstract classes are available but no explicit interface or type definitions. Conceptually, interfaces or type definitions are available. The syntax to specify an interface, however, makes use of an abstract class. C++ supports multiple inheritance, i.e., each class may have several parent classes.

Types, Class vs Interface Inheritance

In contrast, Java permits multiple inheritance only for explicit Java-interfaces. A Java-interface is a special syntax construct: It is a name for a set of abstract methods and constants that is similar to the definition of a type. Java-classes can implement one or many interfaces, implying that the implementing class defines each abstract method of these interfaces. Java-class inheritance is restricted to one parent class, multiple inheritance is only allowed for interfaces. As result, Java language constructs encourage the distinction between class and interface inheritance and the explicit definition of types that are completely independent from implementation.

4.2.4. Grand Redesigns and Choice of Programming Language

Grand Redesigns

"The grand redesign in the sky" (Martin, 2009, pp. 4) is a story about the attempt to replace a production software completely by a redesigned version. While the development of the redesign took place, the development and improvement of the "old", still running

production software kept on. At the end, the old production software was never replaced by the redesign because the redesign never caught up the functionality of the production software. Other stories report that grand redesigns can work[4], but result in much effort.

Choice of Language

Need for design patterns and OO-concepts can be reduced if the programming language and their underlying concepts are chosen carefully (Heinlein, 2007). So, why think about design patterns and other sophisticated OO-concepts when a change of programming language can solve the problem?

Change of Language

The concepts of a new language can be appealing but other requirements and restrictions might exist. Project requirements can comprise availability of IDEs, build management, and continuous integration tools. Available time and skills of developers might also imposes some restrictions and might require initial motivation and training effort. This is fairly important as programming styles typically should not be transfered $1:1$ from language to language. If developers are taking over their programming style for the old language to the new language they might write a huge amount of code that is no longer needed due to improved concepts in the new language. Project specific constraints and requirements may motivate and restrain a change of programming language at the same time. Reimplementation in a new programming language can be considered as hard as a *Grand Redesign*. One might conclude that if a project once is set up it will stick to its initial programming language, forever. A way out of this trap can be a slowly drifting translation to a new language. This gets easier, the more seamless the integration of the programming languages is possible, i.e., the more IDE, build management, and continuous integration tools can be used for two languages that can exchange information without huge effort for migration. Luckily, for the Java world, the number of such tools currently increases with the rise of languages as Groovy or Scala[5].

4.2.5. Coupling and Cohesion

Probably two of the oldest concepts for software architecture are *coupling* and *cohesion* (also cohesiveness) (Stevens et al., 1974). Coupling is a software engineering standard term used to describe the degree and the type of dependency between software elements. Coupling is low when dependencies are few and weak. Cohesion refers to the strength of the relationship between the functions of a software element. High cohesion means that all operations bundled in a software element are required for fulfilling the element's main task and only few other functions exist that do not contribute to this task. For both coupling and cohesion many different attempts to define software metrics have been published. Metrics and software measurements are however out of

[4] e.g., `http://www.wired.com/wiredenterprise/2013/06/facebook-hhvm-saga/3/`, last access 25.07.2013

[5] see `http://groovy.codehaus.org/` and `http://www.scala-lang.org/`, last access 26.07.2013

scope for the project of interest. *Low coupling* and *high cohesion* are used as a more general design goal that is not backed by numerical measurements. Readers are referred to Fowler (2001) for a further introduction to this high level view on the two concepts.

4.2.6. Data and Hybrids

Hybrids

Martin (2009, pp. 93) discusses the differences that may be made between *Objects* and *Data Structures*. Objects expose methods to their user that allow the modification of the hidden data and state within the object. Data structures are, technically speaking, also objects that solely offer access to data. Data structure objects do not hide an internal state and do not have methods.

In certain programming languages, however, also data structures may have methods that are used to access the fields containing the data – so called getters and setters. In Java getters and setters can be added to data structures, e.g., because of the requirement to define interfaces for data structure objects or use of the bean specification that requires a getter and setter for nearly all fields of a bean. If data structures and behavioral objects are not clearly separated a `getFoo()` method can be both: A simple getter for a field value or a method that is returning the desired field value but meanwhile has some side-effects. The latter is considered as bad programming style as the caller of the method might not be aware of the side-effect. Martin (2009, p. 99) calls such structures, that possess properties of data structures and behavioral objects, *Hybrids*. Some functions of a Hybrid define behaviour and modify the object's state while other functions are just accessors to its encapsulated data. Existence of Hybrids complicates the extension of objects' behavior as well as adding new data structures. Overall, Martin (2009, p. 99) concludes that Hybrids should be avoided and *"are indicative of a muddled design"*.

4.3. MATSim in 2007

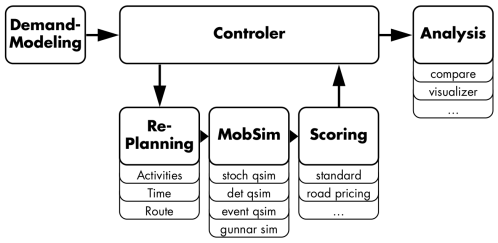
This work was started mid 2007. The first release (MATSim_20070516)[6] of MATSim appeared a few month before. This section summarizes the software architecture of this first release and sketches some resulting problems. In principle, the main steps of the algorithm presented in Sec. 2.1 have been implemented in the first releases but functionality was limited to car traffic only.

2007 State of MATSim

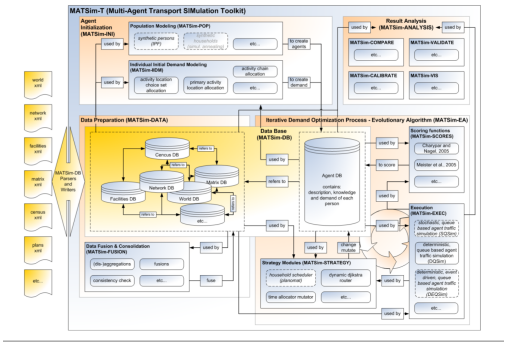
The software architecture in MATSim_20070516 was at a rather prototypical state. It was a reimplementation of the C++ precursor that was developed and used for many studies(e.g. Cetin, 2005; Raney, 2005; Charypar, 2008). First design drafts to improve architecture have been proposed by Rieser (2006) covering thoughts on modularity and setup of control flow, see Fig. 4.1a. Also, first unit tests had been set up and were run

[6] see `http://sourceforge.net/projects/matsim/files/MATSim/`, last access 11-01-2013

over night in a non-standardized, self-scripted procedure (Rieser, 2006). An overview of the software at this time is shown in Fig. 4.1b.



(a) The control flow of MATSim in 2006 (source: Rieser (2006))



(b) Overview of MATSim in 2006 (source: Balmer (2007))

Figure 4.1.: The control flow of MATSim in 2006

In 2007, MATSim had (and still has) the following three major data containers: Data Containers

Network: A collection of links and a collection of nodes, making up the road network.

Population: A collection of all virtual persons, where each person may contain plans (see Sec. 2.1).

Configuration: All information relevant for the configuration of a run.

The developers had made a decision to consider these data containers as singletons, with the reasoning that MATSim was oriented towards large-scale simulation, meaning that it was not expected that a simulation might use two of these containers simultaneously.

They also had made a decision to make them globally accessible by static methods. Access to them could be gained by methods such as `Gbl.getNetwork()`. This approach was displaying several problems sketched in the following.

The central data containers were accessible from everywhere. All methods could, as a side effect, access or even modify the contents of these containers, which was done quite often. The method names usually were not reflecting these side effect. Thus, method names were not trustworthy. To avoid mistakes, one had to check recursively all subroutine calls for side effects. Side Effects

This also meant that methods could only be reliably reused when all containers were filled with meaningful data. Otherwise, it would happen quite often that somewhere in the call hierarchy of a method access to one of the global containers or its content was needed, stopping the execution with a null pointer exception. No Modularity

Further, class inheritance was used to add behavior to the central classes. For exam- Class Inheritance

ple, there was Node as explained above, RouteNode extends Node as data class to store the intermediate results of the Dijkstra algorithm, and MobsimNode extends RouteNode which contained the intersection logic of the traffic flow model. Similarly, there was DriverAgent extends Person, which contained the driving logic. Everything was put into *one* inheritance hierarchy in order to instantiate the objects only once and save both on memory and on time to instantiate objects.

In consequence, it was impossible to replace one object type by another. For example, it was not possible to replace the default driver agent by some specialized driver agent which could do, e.g., within-day replanning. Putting DriverAgent behind an interface (and introducing programmable factories at the right level) would have solved that issue, but there would still have problems with the objects "in the middle" of the hierarchy: For example, replacing RouteNode by an alternative class to try out alternative routing algorithms was nearly impossible in a pluggable way since one would have needed to tell the DriverAgent to now inherit from that alternative class.

Also, functionality was typically added directly into the behavioral object. In consequence, central objects were modified by everybody with repository access, typically without a test case. Given that the central objects did not expose all internals, people had no alternative.

Hybrids

On a more abstract level, the approach meant that central MATSim objects – links, nodes, and persons – were implemented as hybrids, reflecting, as data classes, the material from the input files while on the other hand having lots of internal behavior.

Another result of this approach was also that the objects in the central containers referenced each other: Synthetic persons, which are members of the population container, referred to links, which are members of the network container. These references were resolved during object instantiation, which was done during file parsing. In consequence, the population container could normally not be parsed without having previously parsed a network which would resolve all link references in the population container. As a result, it was not possible to process the population without having a network, meaning that the population part of the code could neither be used, nor maintained, or tested without the network part of the code.

No Uniform Build Process

The build process of the build server and of some users was defined via a make file. In contrast, developers were advised to set up their Java classpath for their integrated development environment (IDE) of choice manually. This implied, that the libraries and dependencies the source code is compiled to and run with, was not specified. The order of manual selection within the graphical user interface of the IDE was specifying the behavior of the executed code. In the worst case, each developer compiled and tested a different software. In order to reduce impact of the problem, a project convention was to use as little externally provided code as possible. This led to the implementation of, e.g., custom logging, coordinate reference system transformations, or "streaming" parsers[7].

[7] Streaming means that data is read from disk processed and written to disk again

All functionality was required and had to be available quickly. In consequence, some of the functionality was implemented at unconventional places. E.g. *Factory Methods* were implemented somewhere in the inheritance hierarchy and were not only used to create objects, but also for XML parsing. The structure shown in Fig. 4.1b was not reflected by the code.

Coupling, Cohesion

Functional extension was ongoing or in planning. In addition to the major three data containers, other components referenced the major containers and were available via globally accessible static methods. In conjunction with the use of class inheritance this led to a high coupling and low cohesion of modules. Reuse of code, e.g., for analysis of simulation results, was hard.

Summary

Summarizing, extensibility was possible, but quite difficult and resulted in complicated type hierarchies. The maintenance costs for the code base were increasing. The next section considers options to improve the situation by use of plain Java.

4.4. Patterns, Java & Extensibility

Patterns & Language

The need for design patterns depends on the programming language. Each language has a set of native language constructs as, e.g., control flow statements. These native constructs often include design patterns. E.g., inheritance is a typical design pattern that comes with all OO programming languages. In pure procedural languages such as C it would be a useful pattern (Gamma et al., 1995). Another example is the Iterator pattern that is available as syntactic construct in most modern OO programming language as, e.g., Java or Ruby. Heinlein (2003, 2007) provides a detailed analysis how well chosen native language constructs can reduce the need for patterns.

Java

Based on the considerations in Sec. 4.2.4, there is neither an alternative programming language in sight which would make programming in the context of MATSim much easier, nor is such a switch of the programming language easily feasible. Thus, the subsequent analysis focuses on the extensibility of Java, its limits, and how design patterns may improve the situation. First, dimensions of extensibility are defined for a simple type hierarchy. Then, an exemplary use case is provided. For the use case two distinct software designs are introduced. The first is really simple and uses no Design Patterns. It is similar to many constructs found in the 2007 release of MATSim. The second design is more complex and reflects the architecture of the current MATSim codebase. The two designs are then extended in all dimensions in order to show how extensibility works out in Java.

Code

Readers that prefer real code to text and code snippets can checkout the examples by executing `git clone https://github.com/dgrether/diss.git` on the command line[8].

[8] the source-code-management system git must be installed on your machine, see `git-scm.com`

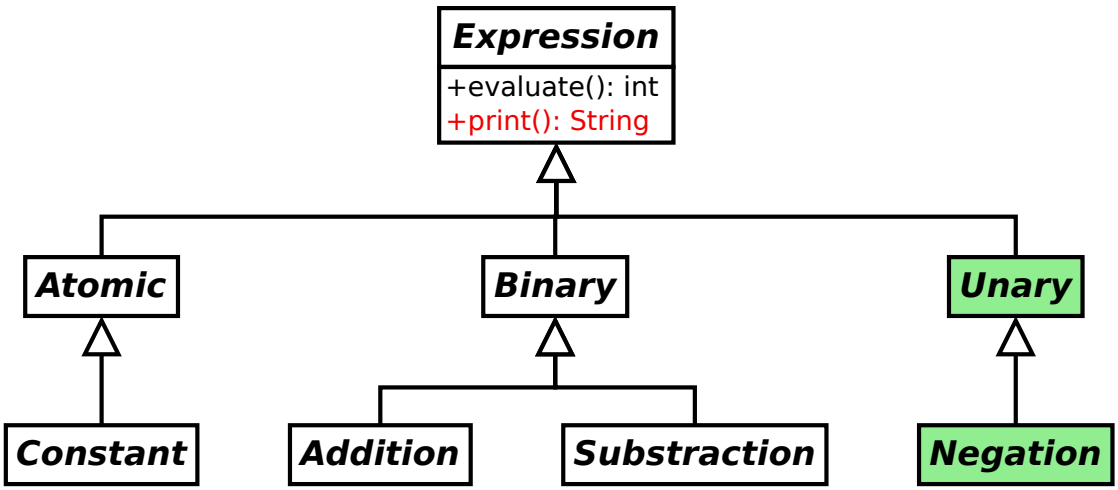


Figure 4.2.: Typical object-oriented type hierarchy. Green: Vertical extension, Red: horizontal extension (source: simplified version from Heinlein (2003))

4.4.1. OO Type Hierarchy – Expressions

OO Extensibility

Heinlein (2003) states that the only available dimension for extensibility that is available in a direct and modular fashion in common OO programming languages is vertical extension. He shows how future programming languages can support the other two dimensions of extension in a more natural and thus user-/programmer-friendly manner. As an example, he uses a simple type hierarchy that represents arithmetic expressions.

Type Hierarchy

Fig. 4.2 shows parts of this type hierarchy. The basic type is `Expression` defining a method `evaluate()` for all subtypes that represent the different elements of basic arithmetic. E.g., the expression $17 + 4$ consists of two objects of type `Constant` that are used as left and right expression of an `Addition` object. Heinlein (2003) distinguishes three dimensions of extensibility in OO programming:

- Vertical extensions: extensions of the type hierarchy, e.g., add a type for Unary Negation (green).
- Behavioural extension: extensions or even restrictive modifications of the original behaviour of operations, e.g., to swap the behavior of the method `evaluate()` in the classes `Addition` and `Subtraction` or to print some debug output to a log stream before evaluation is done.
- Horizontal extensions: extension of the set of operations available on types, e.g., add method print() to Expression (red).

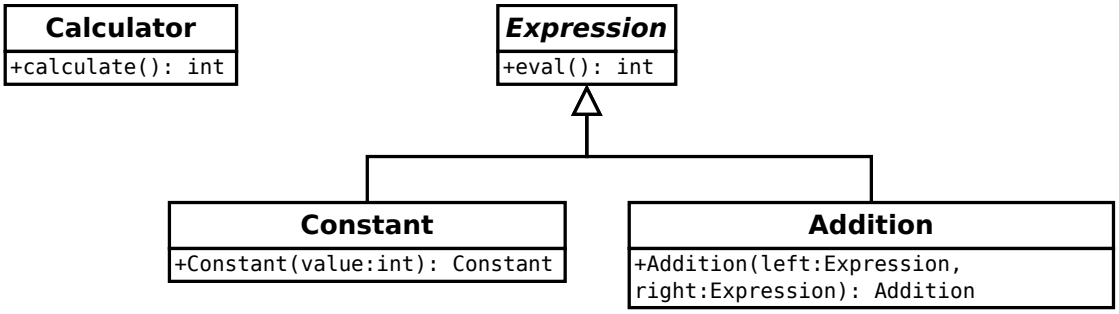


Figure 4.3.: Software design for the simple calculator

4.4.2. Use Case – Calculator

Use Case

A typical use case for the type hierarchy shown in Fig. 4.2 could be a `Calculator` object with a single method `int calculate(..)`. Parts of the program are skipped that deal with parsing and representing the input data. This data is assumed to be available as parameter of `createExpression(..)`, e.g., as data transfer objects (DTO), see Martin (2009, p.100). Suppose the input is `17 + 4`. Then, `Calculator.calculate(17 + 4)` has to create two expressions of type Constant and one expression of type Addition. The left part of the Addition is the Constant representing 17 while the right part is the Constant for 4.

Simplicity?

Indeed, the `Calculator` and the `Expression` hierarchy are simple examples and at a first glance the design seems to be exaggerated. The structure of the example, however, is a minimalistic version of many real-world software architectures. Suppose the `Calculator` is a simplified form of the MATSim `Controler` class. Then, the `Expression` type hierarchy can be seen as a pendant to the classes used to model MATSim's algorithm. As the example and the real-world problem are quite similar a transfer of software design from the example to MATSim is feasible.

In the following, the `Calculator` and the `Expression` type hierarchy are used to explain how the three dimensions of extensibility can be provided. The analysis is restricted to Java as programming language and reveals where, due to the language concepts of Java, extensibility comes to its limits.

Simple Calculator Design

Simple Calculator

A naive, straightforward design for the calculator is shown in Fig. 4.3. A Java-interface `Expression` defines a type for expressions with a single method `eval()` returning an integer. `Expression` is implemented by two classes: `Constant` and `Addition`. Besides the method `eval()` the classes provide public constructors. The method `calculate()` of the `Calculator` class uses this hierarchy, e.g., to calculate the outcome of 17 + 4. The code of `calculate()` is shown in Listing 4.3.

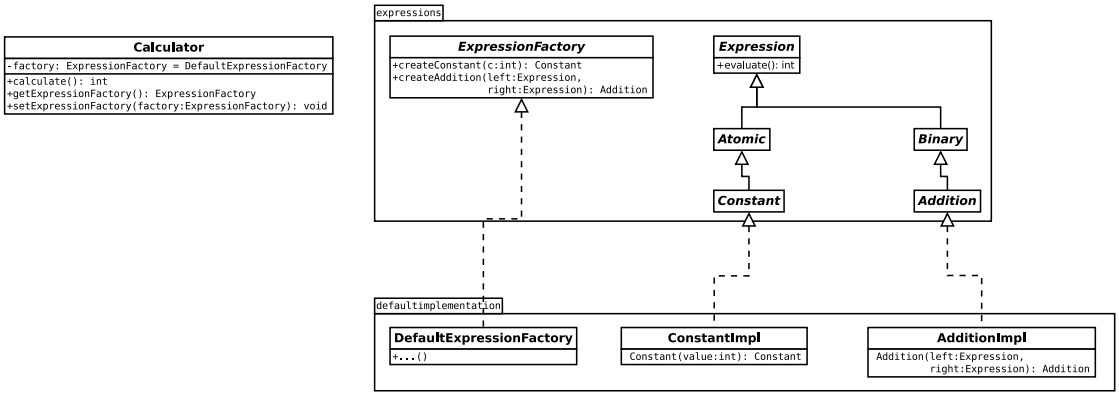


Figure 4.4.: Software design for the advanced calculator

```
public class SimpleCalculator {
  public int calculate(){
    Expression a = new Constant(17);
    Expression b = new Constant(4);
    Expression add  = new Addition(a, b);
    return add.eval();
  }
```

Listing 4.3: Simple calculator, Usage for 17 + 4

Advanced Calculator Design

Advanced Calculator

A more advanced design for the calculator is shown in Fig. 4.4. The type hierarchy for `Expression` is provided as Java-interfaces. In the same package, a Java-interface for a factory with creational methods for `Constant` and `Addition` is specified. In a second package, a default implementation is provided. Implementations for `Constant` and `Addition` are only visible within the package. The only exposed class is the `DefaultExpressionFactory` that must implement all methods of `ExpressionFactory`. The calculator has an attribute that holds the default implementation of `ExpressionFactory` and provides getters and setters for it. The code of the `Calculator` class is shown in Listing 4.4. At a first glance, functionality is the same as for the Simple Calculator. The more complicated design pays off when extensions or modifications of the code have to be implemented.

```
public class ExtensibleCalculator {
  private ExpressionFactory factory =
    new DefaultExpressionFactory();

  public int calculate(){
    Expression a = factory.createConstant(23);
    Expression b = factory.createConstant(19);
```

```
    Expression add  = factory.createAddition(a, b);
    return add.eval();
  }
```

Listing 4.4: Advanced calculator

4.4.3. Extensibility of the Calculator

The two Calculator designs and implementations of the preceding section are extended in all three dimensions.

Vertical extension: Negation

As exemplary vertical extension `Negation` shall be available to the calculator as subtype of `Expression`, see Fig. 4.2. The method `calculate` of the `Calculator` implementation shall now calculate the result of $-(17+4)$.

Listing 4.5 shows a possible approach for the simple design. The code of the `calculate()` method is copied from the simple calculator (see Listing 4.3) and a new line is added that negates the `Addition`. The class `Negation` is added as implementation of the `Expression` Java-interface. Simple Design

```
public class SimpleNegationCalculator extends SimpleCalculator {
  public int calculate(){
    int add = super.calculate();
    Expression c = new Constant(add);
    Negation neg = new Negation(c);
    return neg.eval();
  }
}
```

Listing 4.5: Simple calculator with vertical extension for Negation

In contrast, Listing 4.6 shows the extension for the advanced design. Delegation is used for everything concerning the calculation of $17+4$. The constructor expects a `Calculator` as parameter. The instance passed is used as delegate. The `NegationExpressionFactory` instance is created using the `Calculator`'s factory as delegate for all operations already available while the method `createNegation(Expression e)` is added to the factory. In the `setFactory(..)` method the `NegationExpressionFactory` instance is passed as factory object to the delegate. The method `calculate()` makes use of the delegate `Calculator` to calculate $17+4$. Then it creates a `Negation` from the result and returns its evaluation. Advanced Design

```
public class NegationCalculator {

  private ExpressionFactory factory;
  private ExtensibleCalculator delegate;

  public NegationCalulator(ExtensibleCalculator delegate){
    this.delegate = delegate;
    this.factory = new NegationExpressionFactoryImpl(delegate.
        getFactory());
  }

  public int calculate(){
    int add = this.delegate.calculate();
    Expression c = this.factory.createConstant(add);
    Expression neg = this.factory.createNegation(c);
    return neg.eval();
  }

  public void setFactory(NegationExpressionFactory factory) {
    this.factory = factory;
    this.delegate.setFactory(factory);
  }
}
```

Listing 4.6: Advanced negation calculator

Findings

The native language concepts of Java comprise inheritance. Implementation of vertical extensions is relatively easy. Thus, at this point, the simple implementation is more intuitive. The advanced design pays off when other extensions shall be combined with a vertical extension.

Behavioral extension: Debug

As showcase for a behavioral extension, a debug functionality shall be added to the calculator in addition to the negation extension. All implementations of `Expression` shall call a print function before evaluating itself. The print function sends the expression to a stream before it is evaluated, e.g., DEBUG: −(17+4).

Simple Design

If the simple design is extended, one needs to implement `calculate()` again from scratch. However, to wrap a few lines around the code of the original `SimpleCalculator` as in the vertical extension is no longer sufficient. The behavior of the objects created within `calculate()` must be extended. Listing 4.7 shows a potential reimplementation. Internally, the debug classes could make use of inheritance. E.g., `DebugConstant` extends the `Constant` implementation from the `SimpleCalculator`. Listing 4.8 sketches such an implementation. This ensures that the same code is used for evaluation and only the functionality for debugging needs to be written from scratch. In terms of overall extensibility, this is, however, not helpful.

```
public class SimpleNegationDebugCalculator {
  public int calculate(){
    Expression a = new DebugConstant(17);
    Expression b = new DebugConstant(4);
    Expression add  = new DebugAddition(a, b);
    Expression neg = new DebugNegation(add);
    return neg.eval();
  }
}
```

Listing 4.7: Simple calculator with debug extension

```
public DebugConstant extends Constant {
  public DebugConstant(int c){
    super(c);
  }
  public int eval(){
    this.printToDebugStream(super.getValue());
    return super.eval();
  }
}
```

Listing 4.8: Implementation of DebugConstant for the simple calculator

With the advanced design the debugging functionality can be added to the code for the Calculator or the NegationCalculator. The code is used without any modifications. There is no need to copy existing code. A factory, that creates instances with additional debug functionality, has to be set as shown in Listing 4.9.

Advanced Design

```
  public static void main(String[] args){
    NegationCalculator nc = new NegationCalculator();
    nc.setFactory(new DebugNegationExpressionFactory(nc.getFactory())
        );
    nc.calculate()
  }
```

Listing 4.9: Main for the advanced calculator with debug functionality

This DebugNegationExpressionFactory applies the factory provided by one of the Calculators. It serves as delegate to create Expression instances. Listing 4.10 sketches one of the create methods. This ensures that the evaluation functionality is implemented only once.

```
  public Constant createConstant(int c) {
    Constant cons = this.delegate.createConstant(c);
    return new DebugConstant(cons);
  }
```

Listing 4.10: Snippet from the expression factory implementation for the advanced calculator with debug functionality

Also applying delegation, these instances are wrapped in DebugExpression instances. Before the evaluate() method of the delegate is called, the desired output is generated. An example implementation is sketched in Listing 4.11.

```
public DebugConstant implements Constant {
  public DebugConstant(Constant delegate){
    this.delegate = delegate;
  }
  public int eval(){
    this.printToDebugStream(delegate);
    return this.delegate.eval();
  }
}
```

Listing 4.11: Implementation of DebugConstant for the advanced calculator

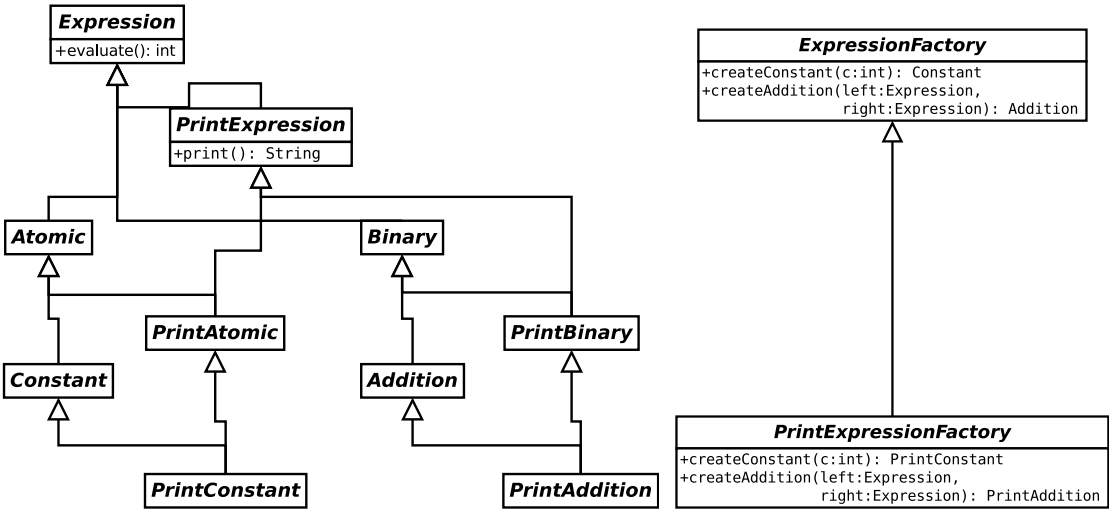
Findings To extend the simple calculator behaviorally, one starts to implement from scratch. The eval() method has not to be reimplemented as it can be inherited via class inheritance. In contrast, the behavioral extension of the advanced calculator only requires a factory. The factory can be applied to a calculator by a customized main() method in that also the concrete calculator implementation is specified. Both calculators of the advanced design, i.e., ExtensibleCalculator or NegationCalculator, can be used with the same factory implementation.

Horizontal extension: Print

API Breaking As an example for a horizontal extension a print() method shall be added to Expression and all subtypes. This implies that all implementations that subtype Expression have to be modified. If all source code is available and modifiable, this can be realized with some effort. Otherwise, the change breaks API – the *Application Programming Interface* that defines access to a software component.

No API Breaking An API break can be avoided. The type definition for Expression is extended by a subtype PrintExpression. The print() method is specified in this subtype. Applying this to all other subtypes of Expression results in the type hierarchy shown in Fig. 4.5a. One can observe, that multiple inheritance is required — at least for the definitions of the types.

Simple Design The simple calculator design uses only one Java-interface, Expression. All other types are defined by classes and do not support multiple inheritance. Thus, to implement the print extension in a non API breaking fashion, some of the methods and classes must be reimplemented from scratch. Similar to the behavioral extension, debug, the calculate () method has to be reimplemented. Within the method the objects implementing the print functionality have to be instantiated. Thereby, the specific calculator is specified and can not be changed without further reimplementation.



(a) Non API breaking type hierarchy for the print extension

(b) Factories for the advanced design

Figure 4.5.: Horizontal extension: print, advanced calculator

When the print extension is implemented, the design of the advanced calculator pays off. The type hierarchy depicted in Fig. 4.5a can be implemented 1 : 1 as Java-interfaces are used consequently. `ExpressionFactory` is extended by a Java-interface `PrintExpressionFactory`, see Fig. 4.5b. The factory methods of this Java-interface return subtypes of `PrintExpression`. Classes that implement the new print Java-interfaces can make use of delegation. An implementation of the `PrintExpressionFactory` can be used with all existing calculators. Code that does not need the print functionality is not affected. Advanced Design

Modules that require the print functionality have to ensure that an instance of `PrintExpressionFactory` is set in the calculator, before any object is created. Then, they can cast the objects of type `Expression` created by any `ExpressionFactory` to `PrintExpression` and use the new `print()` method. One may argue that this cast exhibits some bad smell of code. However, print is an extension to the original calculator. The API of the original calculator never pretended to support a print method for expressions. If the print method is required, one refers to the extended API. The cast expresses this requirement and fails if it is not met.

4.4.4. Discussion & Conclusion

One could conclude that Java code can be extended in all three dimensions if design of the advanced calculator is applied. Supported by the native language construct inheritance, vertical extensions are straightforward. They can be realized with sole class inheritance. Horizontal and behavioral extensions, however, get complicated when class Costs of Extensibility

inheritance is used. The amount of code that needs to be reimplemented from scratch increases with each extension. Use of Abstract Factory, Java-interfaces, and delegation reduces the amount of reimplemented code. Then, also behavioral extensions can be added without much effort. Horizontal extensions, however, seem to be the hardest extension, also for the advanced design. Even in the small example for the calculator, the type hierarchy gets quite complicated after one single horizontal extension. Suppose, a real-world type hierarchy is extended horizontally one or even several times. It is obvious, that the resulting type hierarchy gets hard to understand and difficult to maintain.

Combining Extensions

Use of factories, Java-interfaces, and delegation pays off, if several extensions shall be combined. As shown in the advanced calculator example, a well designed factory can reduce the amount of reimplemented code. In the calculator example, the number of lines is quite small. For real-world applications, however, the number of lines increases quickly. Then, the assembly of software components gets nontransparent and is prone to errors. The problems with horizontal extensions, however, persist.

Visitor

One may argue that for horizontal extensions the visitor pattern can be implemented. Following Gamma et al. (1995, pp. 335), Visitor is worth further consideration if the visited type hierarchy is quite stable — a requirement that is at odds with the desire for other extensions. Among the design patterns in Gamma et al. (1995), Visitor is one of the most complex patterns. Thus, one should consider the implementation of the Visitor pattern carefully.

Prototyping

Without a Visitor implementation, at least one single horizontal extension is feasible, if Java-interfaces are used. This is well suited for prototypes. In particular, if a prototype is a candidate for integration into the main type hierarchy, the advanced design may reduce effort. As long as the prototype is under development, the type hierarchy can be extended similar to the type hierarchy of the print extension. Existing software tests, that are not affected by the extension, can be run for the prototype and the main type hierarchy. The tests have to be rerun with each factory implementation, further changes are not required. Then, if the prototype is selected for integration, one can ensure that previously existing functionality is not affected by the integrated extension — as far as the functionality is covered by software tests.

Conclusion

Overall, the use of Abstract Factory, Java-interfaces, and delegation seems to pay off for a software relying on pure Java. Their application for extension, however, comes with some complexity.

4.5. Dependency Injection & Aspects

The considerations of the last section take concepts into account that rely on the knowledge and use of pure Java. Pure Java has the advantage that language basics and usage are well established and can be taught and learned quickly. At TU Berlin, a 4 hour, one

term class for engineers is sufficient to teach students the basics they need to attend the class that teaches application of MATSim. But, pure Java has some limitations and is conceptually not straightforward as one needs at least delegation, well designed interfaces, and factories to handle extensibility up to a certain limit. Tools that are not standard Java but enable *Dependency Injection* can help, but may increase overhead for getting started. This section discusses if and how Dependency Injection might improve extensibility.

4.5.1. Dependency Injection & Spring

Inversion of Control & Dependency Injection

Dependency Injection is a more specific name for a concept that is also referred as *Inversion of Control (IoC)* (Fowler, 2004a). What does Inversion of Control or Dependency Injection mean? Before a software module can be constructed, the dependencies required by the module must be instantiated. Dependency Injection provides conventions to specify the dependencies of modules apart from their source code. A Dependency Injection mechanism constructs software modules. Thereby, it reads the specification, instantiates the dependencies, and injects them into the module.

In the advanced calculator example of the preceding section a specific factory has to be instantiated before an `Calculator` implementation can be used. This establishes a dependency from the `Calculator` to the `ExpressionFactory`. The dependency was resolved by a custom `main(..)` method for each variant of the calculator. This stops the debug or print functionality being a plugin (Fowler, 2004a).

Spring

The *Spring* framework[9] is an addition to standard Java. Its rich, highly modular, and well documented set of features can reduce programming overhead in Java standalone- and web-application development. At time of writing, the recent release documentation (Johnson et al., 2013) covers topics as, e.g., modules for dependency injection, data access and validation, or a language extension of Java for *Aspect-Oriented Programming*. In the following example, the calculator example is constructed with the *IoC container* of Spring.

4.5.2. Dependency Injected Calculator

Calculator Assembly

The components of the calculator example (Sec. 4.4.2) are assembled with Spring's *Dependency Injection* mechanism[10]. For each extension of the calculator a different `main()` method is required. Therein, the different parts of the `Calculator` are instantiated and plugged together. In the advanced design, this `main()` consists of the following steps — instantiate the `ExpressionFactory` and the `Calculator`, instruct the `Calculator` to use the factory, and compute the result.

[9] see `www.springsource.org/`, last access 14.02.2013

[10] Examples for the calculator in this section can be retrieved via `git clone https://github.com/dgrether/diss.git --branch springbranch` on the command line if git is installed properly.

Potential Spring Assembly

Applying Spring's dependency injection mechanism, the `main()` method do not have to be rewritten for every variant of the calculator. Listing 4.12 shows a potential application of Spring's `ApplicationContext` to push assembly of the calculator to a XML file: `calculator.xml`. The `ApplicationContext` automatically instantiates all modules (*beans*) configured in the XML file and injects their dependencies. Afterwards, from - `ApplicationContext` a completely configured `Calculator` instance can be retrieved to calculate the result.

```
public static void main(String[] args) {
  ApplicationContext context = new FileSystemXmlApplicationContext(
      new String[] {"./src/main/config/calculator.xml"});
  Calculator calc = context.getBean(Calculator.class);
  System.out.println("Result: " + calc.calculate());
}
```

Listing 4.12: main method using spring for the calculator example

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans" ..>
  <bean id="expressionFactory" class="de.dgrether.diss.expressions.defaultimpl.
      DefaultExpressionFactory" />
  <bean id="calculator" class="de.dgrether.diss.AddCalculator">
    <property name="factory" ref="expressionFactory" />
  </bean>
</beans>
```

Listing 4.13: Dependency Injection with spring & XML for the advanced caculator example

An example *Spring* XML configuration for the advanced calculator is shown in Listing 4.13. The first `<bean..>` element states that the `DefaultExpressionFactory` should be available within the container under the id `"expressionFactory"`. The second `<bean..>` element then advices Spring to instantiate an `AddCalculator` and pass the object with id `"expressionFactory"` via `AddCalculator.setFactory(..)` to this `AddCaculator` instance.

Findings

The presented example applies a pure XML configuration to Spring's Dependency Injection. In principle, the XML configuration of each `<bean..>` element can contain any number of NAME $\rightarrow$ VALUE properties. These are set to the instance via appropriate `setNAME(VALUE)` methods. Alternatively, the specification of the dependencies and parameters can be defined via Java code.

4.5.3. Discussion

Example

For the calculator example, the steps to instantiate and run the application can be encoded in four lines of code. Assembly of real software, however, is normally more complicated, the number of lines of code increases rapidly. The more complex the assembly, the higher is the possibility that domain specific code is mixed with code for

component setup and assembly. In consequence code reuse and combination of plugins gets harder.

Dependency Injection

The preceding example illustrates the straightforward use of Dependency Injection. Additional components can be plugged into a core software without touching the core components. There is no need for customized `main(..)` methods. This results in a decoupling of object initialization and production code. If core software and plugins are developed by project contributors that do not work in the same group or do not even share a jointly modified codebase, decoupling is considered very helpful. For plugins, the advantage is obvious. In contrast, the assembly of core software modules via external configuration files and Dependency Injection does not have to ease the development process. A configuration, that is spread over several files, may not be administrable for some projects (Fowler, 2004a). If Dependency Injection for core software proves beneficial depends on the development process, the preferences of team members, and the size of the modules. Thus, an advice cannot be provided on this level of abstraction.

Factories

If component assembly is taken by Spring the Abstract Factory pattern to provide extensions gets dispensable. Both, Abstract Factory and Dependency Injection help to make *"a system independent of how its products are created and composed"* and make a system configurable with *"multiple families of products"*(Gamma et al., 1995, p. 88). However, Dependency Injection cannot enforce independence of how the products are *"represented"*. Neither the relationship between object families that shall be used together is enforced, nor the use of interfaces. Thus, Abstract Factory may still be required in addition to dependency injection. Possibly not on the top level of a software, but for the inner design of complex modules that are target of further development and extension. Then, the abstract factory implementations provided in the calculator example could be replaced by a more intuitive implementation. Their application should be reconsidered, if Java is extended by additional language concepts.

Aspect-Oriented Programming

For the Java programming language exist additions that provide *Aspect-Oriented Programming (AOP)* functionality, e.g., Spring or *AspectJ*[11]. The available features for aspect-oriented programming depend on the addition. With the aspect-oriented addition of Spring, aspects can be applied the same way as in AspectJ, except that only a 80 % subset of AspectJ's functionality is available. Readers that are not familiar with AOP but know design patterns can view aspects as something similar to a Friend or Visitor Pattern implementation that is available as native concept of the programming language. As a seamless addition to Java, aspect orientation offers *"another way of thinking about program structure"* (Johnson et al., 2013, p. 193). Aspect orientation may ease extensibility of Java as horizontal extensions can be realized by aspects. With the new language constructs, however, one would also expect new pitfalls. For a rather small domain, as the transport modeling domain for that MATSim is developed, sorting out opportunities and pitfalls may be overwhelming. Introducing aspect programming may be similarly difficult as changing the programming language (Sec. 4.2.4).

[11] see `eclipse.org/aspectj/`, last access 15.02.2013

4.5.4. Conclusion

Spring for MATSim

Overall, Dependency Injection and other features of Spring could help to ease development of MATSim. Especially for plug-ins, dependency injection is considered helpful. For the core software, this is not obvious and should be subject to further discussion by core developers.

Other Spring features as, e.g., the standardized access to data structures can also help to focus more on the domain, i.e., the modeling of transportation. However, in a community like MATSim where developers are not necessarily computer scientists one could not expect each developer to read 717 pages technical manual before getting started. Instead, a subset of Spring features can be defined. Then, a faster briefing of developers may be provided.

At the time of writing, the developer team[12] is of the opinion that a spring-like configuration of MATSim can also be achieved by making the set-up process of a simulation sufficiently sequential, scriptable, and pluggable. Sorting out the set-up process in such a way would also be a pre-requisite before moving to a framework like Spring. Also, at the point the MATSim components are not considered to be sufficiently general to justify a move to a component framework.

4.6. Software Design of MATSim

In this section, parts of the overall software design for the 2012 version of MATSim are presented. For different components of MATSim potential extensions are discussed.

4.6.1. Overall Architecture

MATSim is implemented in pure Java, frameworks like spring or language additions as AspectJ are not used.

Build Process

Build processes for Java software can be managed by Apache Maven[13]. Maven resolves the problem concerning the build process and encourages developers to use other open source libraries instead of reimplementing functionality that is clearly not within the domain of transport engineering. Furthermore, it allows the convenient setup of standard continuous integration tools and provides standardized builds.

Layered Architectures

The overall architecture of the first release of MATSim in 2007 was rather fuzzy. Coupling was high and modularization at a preliminary stage. Complex code inheritance hierarchies and global variables persisting the state of modules reduced Reusability and

[12] e.g., see `https://matsim.atlassian.net/browse/MATSIM-142`, last access 27.11.2013
[13] see `maven.apache.org`, last access 16.01.2013

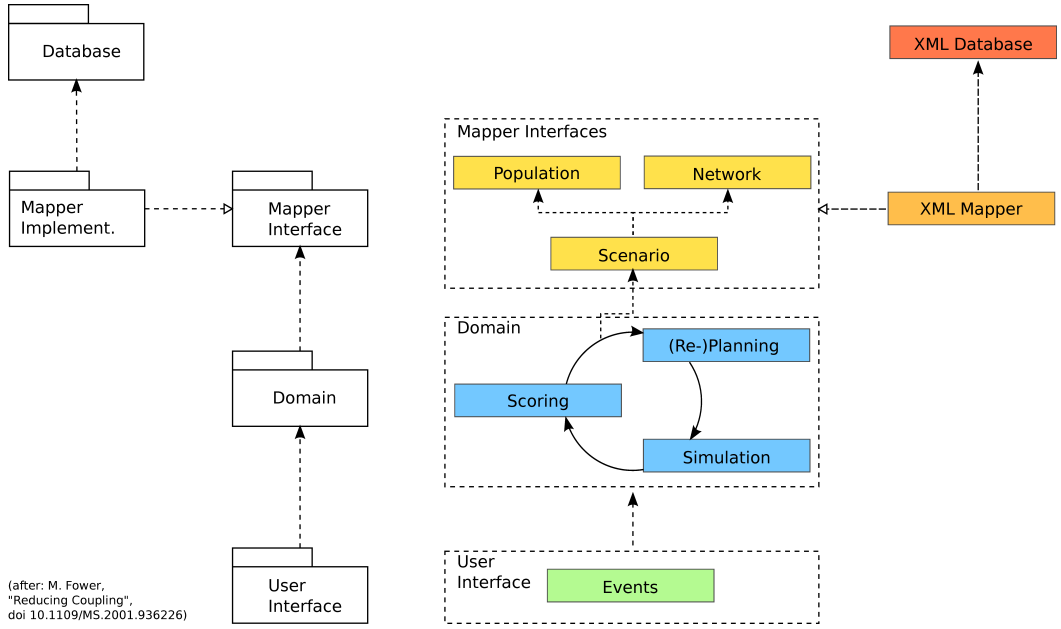


Figure 4.6.: Layered architectures, left: general layer structure, right: layers of MATSim

Extensibility. Especially Hybrids, i.e., objects that possess properties of data structures and provide functionality, reduced Modularity.

Layered architectures, *"that should be familiar to anyone who works with information systems"* (Fowler, 2001) can help to reduce coupling, improve modularization and sort out dependencies between modules. The typical overall layer structure is shown on the left side of Fig. 4.6. It consists of a Database, a Mapper implementation accessing the concrete Database, an abstract interface for Mappers, the Domain to be modeled in the software, and a User interface.

Layered MATSim

This typical layered approach is applied to MATSim. The right side of Fig. 4.6 shows a colored version of the control flow of MATSim that is used for documentation in many sources. In yellow, the mapper to the Database is depicted, blue elements represent the main modules of the Domain while green is the User interface layer.

Design Patterns

Motivated by the considerations in Sec. 4.4, the software design for MATSim uses mainly two design patterns from Gamma et al. (1995). First, Abstract Factories are provided that construct the central components of the software and force use of Java-interfaces instead of implementations. Second, in order to make control flow and object communication transparent, the Observer pattern is applied frequently.

4.6.2. Database & Mapper Layer

Database

Microscopic transport simulation can potentially make use of lots of data. None of the

institutions participating in the development focusses on data modeling, nor is expert knowledge for modeling data ubiquitous at the developer side. Due to this consideration, the amount of data structures required by the core algorithm is reduced to a minimum: An abstract graph representation of a transport network, and a population of virtual persons that are travelling within the transport network. For both data structures, corresponding mapper interfaces and implementations are provided. These can be added to a mapper container, the `Scenario`. Additionally, access to the mapper for the configuration is provided. Within the iteration cycle, access to the data is granted via `Scenario`.

Extensibility

For many studies, more data is needed than provided by network and population. Data sources are heterogeneous and may vary from study to study. In consequence, a manually programmed simulation logic must be encoded against specific concepts derived from data. Effort of implementation is reduced if these concepts can be accessed in a type-safe manner. Maintenance cost increase significantly and extensibility is reduced if central parts of the central algorithm need to parse Strings that have to be well formed. Thus, modules that extend the core simulation functionality are expected to implement their own data mappers. These can be added to `Scenario` as shown in Listing 4.14[14].

```
public void addScenarioElement(Object o);
public boolean removeScenarioElement(Object o);
public <T> T getScenarioElement(Class<? extends T> klass);
```

Listing 4.14: Extension of `Scenario`: Signature

The Signature of the `getScenarioElement(..)` method may look a bit scary, but usage is simple, as shown in Listing 4.15: A custom data mapper of type `MyData` is added to an instance of `Scenario`. Later, it is retrieved via the `getScenarioElement(..)` method in a type-safe manner.

```
MyData data = new MyData();
scenario.addScenarioElement(data);
...
MyData data2 = scenario.getScenarioElement(MyData.class);
```

Listing 4.15: Extension of `Scenario`: Usage

4.6.3. Domain: Simulation, Scoring and Replanning

Core Modules

The central algorithm of MATSim consists of three steps: Simulation, Scoring and Replanning. As indicated by Fig. 4.6, each step is associated with a separate module. In 2007, the functionality of these modules was rather limited, while the number of users willing to extend the software was growing fast. The standard way to enhance or replace those modules was to extend a central `Controler` class and to over-ride the

[14] Note, that the syntax in the current, 2013 version, was slightly changed, see `https://matsim.atlassian.net/browse/MATSIM-68`, last access 27.11.2013

relevant methods that would configure or run each of those steps. This lead to the following problems:

- Extensions, which were developed in this way, were normally not "orthogonal", i.e., one could rarely use both extended functionalities simultaneously. Such orthogonality, however, was clearly a core interest in order to make MATSim capable to address more and more transportation planning problems.
- The `Controler` was not specifically designed for inheritance. In particular, the control flow was distributed over many methods, and each of them could be overridden. In consequence, changes to the control flow could easily break derived classes, for example by moving a control statement to a method that was overridden (and thus ignored) by a downstream users.

Thus, a simple method for functional extension of these modules was required. The control flow and communication between these modules, however, should become invisible to enforce modularization.

Interfaces & Factories

In order to hide implementation details of the core modules, Java-interfaces are extracted from the concrete implementations of the 2007 version. Use of these interfaces is encouraged by Abstract Factories creating the modules. On this top-level view on the domain, interfaces can be rather abstract. E.g., Listing 4.16 shows the top-level interface that a mobility simulation needs to implement in order to work within MATSim, while Listing 4.17 shows the appropriate interface of the factory creating the mobility simulation.

```
public interface Mobsim {
  public void run();
}
```

Listing 4.16: Top-level Java-interface for a mobility simulation

```
public interface MobsimFactory {
  public Mobsim createMobsim(Scenario sc, EventsManager eventsManager);
}
```

Listing 4.17: Top-level factory to create a mobility simulation

This enables users to replace the mobility simulation completely if their requirements are not met by the default implementation.

Observer

In order to hide control flow and communication between modules, the Observer pattern is used at two places:

- First, the mobility simulation uses the same Observer implementation for `Events` as found in the 2007 release. The mobility simulation retrieves access to an `Events Manager` when it is created via the factory method, see Listing 4.17.

- Second, new functionality to the steps of the central algorithm can also be added via an Observer: The state of the iterative relaxation process represents the Subject of the pattern. States are, e.g., startup and initialization completed, the start of an iteration, the end of microsimulation execution, the end of the scoring or the end of the iteration cycle[15]. The central algorithm of MATSim alternates the states of the subject (currently implemented in the `Controler` class). Users can provide custom implementations for Observers. This allows addition of functionality at each point of the iterative process. E.g., code for analysis could be added at the end of each iteration without touching any code concerning setup and control flow of the overall simulation process.

Findings

The combination of Abstract Factory and Observer makes the central algorithm extensible. Users can connect to the core algorithm at each step and add their specific Observer implementation that may provide any required additional functionality. If the core modules do not fit needs, they can be replaced completely. Thus, the overall process is considered to be flexible and extensible. However, replacing a whole module of the core algorithm, as e.g., a complete replacement of the mobility simulation, can result in much effort. Therefore, default models and implementations are required, which are also suited for extension and customization. These default models require interface definitions to provide extensions access to their functionality.

4.6.4. Default Models & Implementations

Defaults

Obviously, specifications of abstract interfaces cannot be run as there is no implementation. Thus, there is a certain amount of default implementations that can be used and configured via the configuration file of MATSim. If a user states he uses MATSim, he should refer to the default implementation. His specific set of configuration options should be documented, if non-default parameters are used. If users add or modify code, it is expected that code modifications are also documented and published. In order to reduce documentation effort and amount of code to be rewritten, certain parts of the default implementations may be reused. To ease effort default implementations are hidden behind abstract type specifications suitable for customization.

Matryoshka Dolls

If Delegation and the patterns Abstract Factory and Observer are used consequently, extensibility can be realized up to a certain extent. In addition, the resulting software architecture can be understood by knowledge of the two patterns and the concept of delegation. E.g., the default production mobility simulation of TU Berlin (referred as `QSim`) uses the Observer pattern twice: First, the `Event` mechanism can be used by Observers to get informed about things that happen in the simulation. A second Observer implementation is provided for the control flow of the mobility simulation. E.g., one can register an Observer that is informed when the mobility simulation is set up, incremented, or shut down. By use of a Factory creation of objects for virtual persons

[15] see `www.matsim.org/node/602` for the official documentation, last access 17.01.2013

or other agents fed into the mobility simulation can be changed. Default behaviour of virtual persons thus can be customized. This repetition of concepts and patterns results in a Matryoshka Doll. Each time one of the modules of overall simulation is unfold, same concepts appear. Thus, the overhead to understand design concepts required for extension and modification is reduced.

Horizontal Extensions

Provided that existing source code should not be touched functional extension is possible by use of patterns as Abstract Factory and Observer in conjunction with the language concepts of Java. However, extensibility is also restricted by this set of programming concepts. Instead of extending existing types horizontally, new operations can also be added as completely separated modules. Then, the default modules require some interface definitions to communicate with the extensions. The Observer implementations provide this interfaces. The next section shows the design of such an extension for a module to simulate traffic signal control.

4.7. Traffic Signals Extension

Traffic signal control is an important element of urban transport systems. For the purpose of transport forecasting and simulation, the inclusion of traffic signal control may be desired, but because of the need for extensive amounts of data, it may not be feasible.

Software Design

In the following, the design and implementation of a software architecture is proposed that allows the modeling of traffic signal control as an optional component of MATSim. Software design and implementation follow the design patterns used in the core of MATSim. Instead of being modeled as extensions to the MATSim type hierarchy, the model is a stand-alone extension that is coupled to MATSim via one concrete Java-interface declaration that has to be implemented by the mobility simulation. It was possible to solve all other coupling by the Abstract Factory or Observer implementations reviewed in the previous sections.

Components

The traffic signal module for MATSim consists of three components: A data mapper providing access to a database, a component that plugs the module into MATSim, and a default implementation for fixed-time control. All components can ease the implementation of further models for traffic signal.

4.7.1. Data

For microscopic simulation of traffic signals, minimally a description of the real-world traffic signals is needed, i.e., at which locations in the network traffic signals are located and which links or turning moves of the network are influenced by them. Individual traffic signals are often assigned to groups; each traffic signal of a group shows the same color. In addition, the data format should allow to specify the control strategy

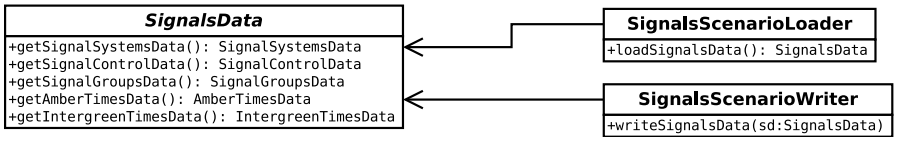


Figure 4.7.: Data layer of the module for traffic signals

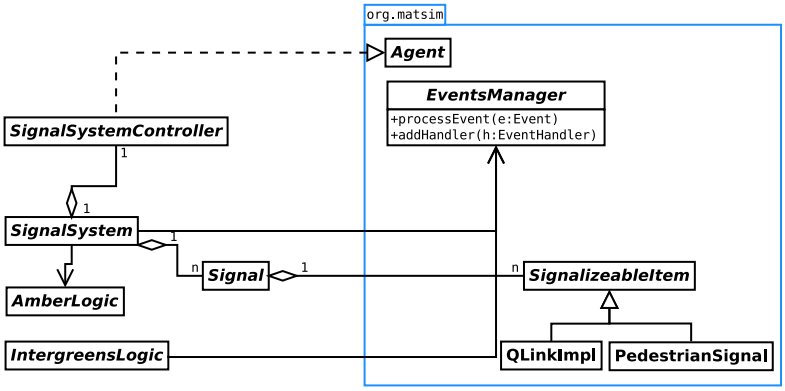


Figure 4.8.: Components of the default module for traffic signals and their connection to MATSim

used for sets of groups. Optionally, one should be able to specify legal constraints. This includes the time of amber or red-amber a traffic signal needs to show before it can switch to red or green. Intergreens specify the time signals have to be all red before a signal can switch to green. All this data is provided by a mapper interface to a database called `SignalsData`. The design of the data mapper is sketched in Fig. 4.7. `SignalsData` provides access to the subcomponents. The mapper is added to the `Scenario` and thus is available at relevant places within the simulation process. The `SignalsData` mapper and its child containers are behind interfaces to ensure that the current XML Schema based data format can be replaced by something else, e.g., a database or web-service.

4.7.2. Default Model & Implementation

Components

Several components are identified that can be used to model and microsimulate traffic signals. Fig. 4.8 shows types of the MATSim core modules from the package `org.matsim` (blue). Furthermore, main types of the traffic signal extension are shown with their dependencies to the core modules. These components and dependencies are reviewed in the following.

Connection to Microsimulation

First, a component that represents the real-world infrastructure is required. It serves as interface to a mobility simulation, i.e., it communicates the color of traffic signals to links of the transport network. The communication is realized by the Java-interfaces

`Signal` and `SignalizeableItem` of Fig. 4.8. The latter is the Java-interface within `org.matsim`. It has to be implemented by mobility simulations in order to support traffic signals. Two currently available implementations are `QLinkImpl` and `PedestrianSignal`.

Control Strategy

The second required component provides traffic signal control strategies for signalized intersections. For real-world applications, it should be possible to control each signalized intersection with a different control strategy. For this, the `SignalSystemController` Java-interface of Fig. 4.8 can be used. This component can be seen as agent if an agent-paradigm is applied. Via the `SignalSystem`, the control strategy can access and switch the state of the infrastructure representation, i.e., the `Signal` instances. Furthermore, access to the `EventsManager` instance of `org.matsim` is provided in order to feed changes of signalization back to the central `Events`' Observer implementation.

Amber

The third component provides a tool for control strategy implementations that reduces the amount of code to switch a signal group from red to green. In between, colors red-amber and amber may be shown for a certain time. This time depends on legal constraints which, in turn, may depend on the layout of the crossing or on the speed-limit. The `AmberLogic` Java-interface is accessible via `SignalSystem`. The logic receives requests to change color between red and green and translates them into a color sequence that shows also the required time of red-amber and amber light.

Intergreens

The last component is a logic that checks if conflicting approaches of a junction got a right-of-way at the same time, producing a warning or an error in such a case. This component is necessary since none of the current mobility simulations available within `org.matsim` represents vehicle collisions. Thus, the component is quite important to ensure validity of the control strategy. The `IntergreensLogic` in Fig. 4.8 provides this functionality. It is a View on the `EventsManager` of `org.matsim` and thus completely decoupled from other components of the traffic signal extension.

4.7.3. Integration into MATSim

Observer

The Observer implementation of the top-level simulation process is used to plug the traffic signal extension into MATSim. After notification that the simulation is at start-up, the signals data is loaded. At the beginning of each iteration, the simulation model for traffic signal control is set up. At the end of simulation some data is persisted. The Observer for the control flow of the mobility simulation is used to update the state of the traffic signal control during execution of mobility simulation.

Abstract Factory

As traffic signals are considered important but still optional for transport simulations, a central factory at the top level of the domain layer is added. Using this factory, the complete module can be exchanged easily, or the default implementation can be retrieved for further customization of its components. Each component can be exchanged or customized via factories.

4.8. Discussion

4.8.1. Overall Design of MATSim

Java

Java as programming language still seems to pay off. Due to its static typing and excellent IDE support, even unexperienced programmers can get their work with MATSim done quickly. In particular, preparation of data for simulation scenarios and customized analysis can be done with little knowledge of Java.

Build Process

The application of Apache Maven for MATSim seems a good choice to build Java applications. At time of writing, Maven got one of the standard build management tools for Java applications and is well integrated into all major IDEs. Due to Maven's dependency resolution problems concerning the build path can still occur but no longer accidentally. Employment of other open source libraries supplying non domain functionality is straightforward and encouraged. Non domain problems can be solved with little programming effort. E.g., logging is no longer implemented within MATSim but handled by Log4J, one of the standard libraries for logging in Java. A wide range of options to customize Log4J is available, more than ever implemented in the 2007 version of MATSim.

Singletons, Hybrids & Layers

All global singletons are removed completely, hybrids are fading. Extensions to standard input data can be added to the `Scenario`. The mechanism described in this chapter can also be found in the API of spring's `ApplicationContext`. Thus, this appears to be standard functionality for extensible Java software. The layered architecture discourages creation of new Hybrids. Extension and customization of data, however, is still is possible. The layers help to separate data from core functionality.

Extensibility

If problems get more complex and MATSim shall be extended, Java has some limitations due to its pure static typed OO nature. Most of these problems can be solved by knowledge and use of the two Design Patterns — Abstract Factory and Observer. The clear limits of the approach are illustrated in Sec. 4.4. Vertical and behavioral extensions can be implemented with Abstract Factory, Delegation, and Java-interfaces. Horizontal extensions, however, are hard to realize and result in somehow messy type hierarchies. Thus, horizontal extensions require some other solution. The Visitor pattern or aspect-oriented programming are considered inapplicable in respect to other constraints for the software. But, horizontal extensions can be replaced by extensions that use the Observer implementations. Instead of adding new methods to the MATSim core type hierarchy, the functionality can be implemented in completely new components. These components can be (de-)coupled to MATSim via the Observer pattern implementations. One could argue that this reduces cohesion as functionality, that belongs together, is not modeled by the same component. On the other hand, one could also state that this increases cohesion on the long run as the transportation domain is structured and decoupled encouraged by the design of the software. At the beginning, cohesion of an extension connected as Observer to MATSim may be low. Over time,

when additional functionality is added to the extension, cohesion may increase while coupling is still at a low level.

Pattern Selection

Software Design Patterns should be selected carefully. If the full spectrum of patterns collected in Gamma et al. (1995) is applied, expert knowledge is required to understand the software. The two patterns, Abstract Factory and Observer, can be understood with relative ease, especially compared to other patterns, as, e.g., Visitor. They are the standard solution for many problems and are among the best known and most frequently used patterns in OO languages.

Modularity

The patterns encourage use of interfaces instead of implementations. The existence of a Java-interface communicates a certain level of design thoughts for a module. The mental barrier to jump over the hurdle to change an Java-interface seems to be higher than just to change some implementation. With the use of Java-interfaces it is more likely that a new module is added instead of inflating existing modules by not cohesive functionality. Thus, the development towards principal software design goals and especially Modularity is encouraged.

Limits of Extensibility

The chapter shows how extensibility can be provided up to a certain extent without changing core components. But the approach is limited. Code applying Abstract Factory and delegation gets hard to understand if several modules are interlaced and used together. The rise of complexity, however, is not a result of the proposed approach. Rather, the complexity is the outcome of the joint usage of several components with interlacing functionality that is hidden by one or more layers of abstraction.

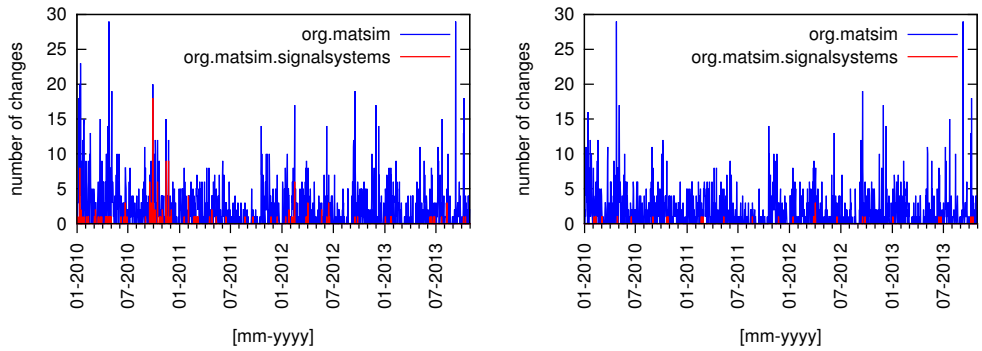
The Observer pattern lacks the definition of the sequence which Observer implementations are informed about changes of the Subject. This might be seen as drawback, but is actually the quintessence of Observer pattern — encapsulating functionality in modules that can be used independently. Thereby, control flow between the modules is reduced to a minimum. The subject informs its observer in a not defined sequence about changes, i.e., *"you don't want these objects tightly coupled"* (Gamma et al., 1995, p. 294). This strong decoupling results in plug-ins that do not depend on each other. Completely independent plug-ins are advantageous compared to the monolithic structure MATSim had in 2007. This limitation of the approach could be resolved in further steps, e.g., by use of spring.

Spring

The transition of the current design to spring's dependency injection could be realized with relatively little effort. Mainly, some top-level factories are no longer needed. Though springs dependency injection must be learned by developers, overall complexity is reduced further. Spring provides a standardized definition for plug-ins. In the current state of MATSim each plug-in is configured differently. Spring's dependency injection mechanism could further improve the plug-in architecture.

Overall

Overall, still not all work is done. In comparison to the 2007 release, however, the software design of MATSim was improved. MATSim is a living project with growing user and developer community.



(a) Changes per day

(b) Changes per day, without some bug-fixes

Figure 4.9.: Changes on package org.matsim and org.matsim.signalsystem

4.8.2. Traffic Signals Extension

Low Coupling

MATSim now comes with a traffic signal module containing a default implementation for fixed-time traffic signal control. The chapter explains how the module is attached to MATSim with the current software design. The architecture of the module follows the same concepts as the core software in order to reduce complexity. The module requires the mobility simulation of MATSim to implement a single Java-interface (`SignalizeableItem`). All other communication between the MATSim core and the traffic signal module can be realized by use of the different Observer implementations. Thus, the extension features a high cohesion while coupling is low.

Extensibility

All components of the traffic signal module can be customized, replaced, or extended without changing the default implementation or core components of MATSim. Due to the use of Java-interfaces, delegation, and Abstract Factory each component can be exchanged separately. The default traffic signal module provides an implementation for fixed-time traffic signal control. For Chapter 5 an extension modeling a traffic-responsive signal control is added by solely implementing the algorithm for traffic-responsive control and the code for the assembly of the components. For the representation of real-world infrastructure, computation of amber times, and coupling to MATSim the code of the default implementation is reused and not rewritten from scratch. Thus, the chosen architecture provides reusable software components and seems to have the required flexibility for extensions.

Stability & Maintenance

Maintenance of the implementation was quite cheap during the last years. Fig. 4.9a shows in blue the changes recorded by the version control system (subversion) to all core code of MATSim within the package `org.matsim` between 2010 and 2012. Changes to the package `org.matsim.signalsystems`, which contains the model specification and the fixed-time control implementation, are shown in red. While `org.matsim` in total was changed quite frequently, changes to `org.matsim.signalsystems` are rare. This affirms

the low coupling between `org.matsim.signalsystems` and the rest of `org.matsim`. The changes to the traffic signal module occurred mainly due to feature improvements and bug fixes by the developer responsible for the extension. E.g., during the period 2010–2012 a feature for visualization was added to the package, also the intergreen logic was developed. Fig. 4.9b shows same information as Fig. 4.9a, but removes all changes due to these bug fixes and improvements. Red changes nearly disappear, confirming the argument that coupling is low.

Overall, this shows that the chosen approach is modular, reusable, uses few patterns, but is extensible.

4.9. Findings

This chapter explains parts of the methodology chosen for the software design of the Multi-Agent Transport Simulation, MATSim. The design focusses on standard architectures and Design Patterns to ease Usability and improve Extensibility of the software. Several design options to improve Extensibility of Java are discussed on a small example. Based on the resulting subset of design principles, patterns, and tools, the software design of MATSim is explained. It is shown that an extension for traffic signal control can be attached to MATSim using this subset. As both, MATSim and the extension, apply the same reduced subset of concepts, the approach can be understood with little effort. Reduction of concepts leads to a clear and in terms of a pure Java software straightforward architecture. As the approach enforces Modularity and Abstraction overhead of maintenance is reduced while Reusability and Extensibility are provided.

Chapter 5

Network Effects of Traffic Signal Control

Overview

The simulation of network wide effects caused by a change of traffic signal control is subject of this chapter. First, some backgrounds are provided. These serve as motivation for a first study, that explains how traffic signals may interact with traffic patterns (Sec. 5.1). Then, based on real-world data, a large-scale scenario is set up (Sec. 5.2). Sec. 5.3 provides a case study for this scenario that illustrates, how traffic-responsive signal control may help in certain situations. Afterwards, in Sec. 5.4, first results of an ongoing research project are presented that aims at optimization and network wide analysis of traffic signal control.

Please note that the case study provided in Sec. 5.3 uses material from Grether et al. (2011).

5.1. Backgrounds and Illustrative Example

5.1.1. Motivation

Signals & Choice

On the short-run interaction of traffic signals and travelers is obvious — staying legal, travelers have to stop in front of a red light. Under consideration of longer time horizons, at least travelers' route and departure time choice may be influenced by the implemented traffic signal control strategy. If one route, between two locations A and B, is considerably faster than other options, one can leave location A later arriving at the same time at B if the fast route is chosen. The speed advantage may no longer hold if all travelers are aware of the advantage. An interaction between the route choice of travelers and the implemented signal control strategy evolves.

Time Scales

This interaction can happen on different time scales and is considered rather indirect — as long as traffic signals are not communicating in real time with routing devices or cars (e.g. Braun et al., 2009). Waiting in front of a red traffic signal is an inevitable

penalty that has to be paid after route choice is made. Learning these penalties might happen within-day, day-to-day, monthly or yearly. This is in contrast to congestion pricing methods. Feasible congestion pricing schemes should communicate the amount of toll for a road to travelers before they enter the road. Thus, traffic signals are a rather indirect method for traffic control.

Models

Traffic signal control can improve overall network performance. Many different control strategies promise improvements. Models that capture the interaction between signal control and travelers' choices can be used diversely – to improve traffic signal control or to predict the outcome of the traffic system if such a strategy is implemented. The latter is the aim of this work.

Traffic Signal Control

Optimization of Fixed-Time Control

Fixed-time traffic signal control periodically assigns a well-defined green-split for each approach of a junction. Traditionally, for optimization of fixed-time signals different regimes of equilibrium traffic flow are determined for several periods of time, e.g., weekday morning, midday, evening and night plus a separate estimate for weekends. These traffic flows serve as input for optimization (e.g. Webster, 1961; Allsop, 1972, 1991; Robertson, 1969).

Traffic-Actuated Control

With upcoming availability of sensors and computer technology these optimizations provided the basis for traffic-actuated signal control strategies. Based on detector input fixed-time signal parameters as green times, cycle, and offsets are adjusted to current traffic situations on the fly. Some actuated approaches use logical operators and functions to adjust signal timings (Friedrich, 2002). More advanced methods as, e.g., SCOOT (Hunt et al., 1981; Robertson and Bretherton, 1991; Bretherton et al., 2004), MOTION (Bielefeldt and Busch, 1994; Busch and Kruse, 2001; Brilon et al., 2009), or BALANCE (GEVAS software Systementwicklung und Verkehrsinformatik GmbH, 2011; Braun et al., 2009) use macro- and mesoscopic traffic models to predict effects of adjustments of signal timings for a certain time horizon.

Adaptive Control

Recent approaches for traffic signal control no longer need a fixed-time control that is adjusted. Instead, the signal program is build completely on-the-fly based on sensor information. These methods originate from different areas of science. One finds rather conceptual studies and methods that can and are used in practice. An example for the latter is TUC (traffic-responsive urban control) (Diakaki et al., 2002; Christina Diakaki et al., 2003; Kraus et al., 2010; Aboudolas et al., 2010; Kouvelas et al., 2011) a control theoretic strategy that uses a linear-quadratic regulator approach to control green splits based on a store-and-forward model of urban traffic. Due its polynomial complexity TUC can be used in real time monitoring the whole transport network. Optional extensions to TUC provide cycle time adjustments, offset optimization, and public transit priority. Also practice ready is the approach proposed by Lämmer (2007); Lämmer and Helbing (2008, 2010). In undersaturated traffic conditions a priority based optimization

of scheduling minimizes local waiting times. A second module stabilizes the optimization when traffic density increases. Green waves are established locally by a prediction model for future arrivals. Rather conceptual is the approach proposed by Cools et al. (2007); Gershenson and Rosenblueth (2009) that looks at traffic as a self-organizing system (Elmenreich et al., 2009). Traffic signals can be controlled by a set of simple rules, coordination evolves from the interaction between cars and signals. Sensor information, however, can be erroneous as some detectors may not work at all or provide incorrect data. If sensor data is erroneous, quality of an adaptive signal control may drop. This problem is addressed by Oertel and Wagner (2011). Advanced traffic detection technologies as, e.g., GPS data or video processing, measure the delay imposed to individual vehicles approaching a signal. When the delay is below a certain threshold a queue clearing policy terminates a green phase. Noteworthy, in simulation studies, the signal control outperforms other approaches when only a part of the individual vehicle delays can be detected.

Agents

Intelligent agents (Russel and Norvig, 2010) can be used to control traffic signals of one or several junctions (Bazzan, 2005). Reinforcement learning techniques enable the agents to control traffic flows (Bazzan, 2005; Bazzan et al., 2010b; Bazzan, 2009). Besides agent-based approaches, signalized junctions can be controlled by other techniques as autonomic and organic computing (Prothmann et al., 2010).

Models for Traffic Signal Control

Equilibrium?

Using Wardrop equilibrium assumptions and continuous link costs, Smith (1979a) shows the existence of a unique and stable static traffic equilibrium. This provides the underlying theory to prove, that traffic signal control influences route choice (Smith, 1979b). The existence of a dynamic traffic equilibrium is studied by Smith (1993). Smith and van Vuren (1993) propose an optimization algorithm for dynamic assignments that takes traffic-responsive signal control into account. The dynamic formulations do not consider the physical extent of queues, they are built up on point-queues.

Combined Problem

The combined modeling of the traffic signal control and traffic assignment is subject of several other research lines. Meneguzzer (1997) reviews their initial roots and defines the *combined traffic assignment and control problem* (CTAC) as finding a tuple (f^*, g^*) of traffic flows f and signal settings g under policy P that fulfills

$$f^* = f^e[g^P(f^*)] \text{ or equivalently } g^* = g^P[f^e(g^*)]$$

where f^e is a function mapping signal settings to equilibrium traffic flows and g^P a function mapping from traffic flows to signal settings under policy P. Nicely, the formulation shows the mutual interaction of traffic patterns and signal settings. A similar problem formulation is given by Cascetta et al. (2006), studying the difference between global and local signal optimization. The time horizon, on that these interactions take place, is not captured by the formulations.

Game theory

Solution techniques for game theoretical models and mathematical programs with equilibrium constraints are similar, especially for bi-level, Stackelberg games (Hollander and Prashker, 2006). Game theory provides techniques and terminology to describe the social dilemma of self-interest vs cooperation between distinct players. For transportation systems game theory provides *"a framework for modeling interactions between groups of decision makers when individual actions jointly determine the outcome"* (Fisk, 1984). The definition of a "leader" is required to model the decisions of a transport authority and the response of traffic participants as Stackelberg game. Mathematical programs also comprise such assumptions but they often are intrinsic to the model and not discussed. A broad review of games used to model and analyze transport systems was undertaken by Hollander and Prashker (2006). Classical game theory is static an explicit representation of time is missing. In most cases traffic flows are modeled statically.

Space & Time

Under the assumption that interactions between travelers and traffic signals over space and time have nicely behaved mathematical properties Hu and Mahmassani (1997) use DYNASMART[1] for simulation. Traffic signal control and travelers' reactions are studied in a day-to-day and real-time context. Traffic-responsive control may improve throughput of a traffic network. The peak load of the network stays equal. The travelers use the improvements to reduce schedule delay (Hu and Mahmassani, 1997). Burghout and Wahlstedt (2007) build a hybrid simulation by a combination of a microscopic and a mesoscopic traffic simulation model. The replacement of a fixed-time control by a traffic-actuated signal control at two junctions changes the travel patterns in the city-wide network.

Dynamics & Game Theory

Game-theoretic formulations for the dynamic interactions between traffic control and traffic assignment are given by Chen and Ben-Akiva (1998). Depending on the type of game used to model the problem, the quality of the solution with respect to the objective function differs. The solution of a monopoly game represents the optimal solution that can be used as benchmark. A Stackelberg equilibrium is superior to the Cournot-Nash equilibrium as user reactions are anticipated (Chen and Ben-Akiva, 1998)[2].

Mutual effects between road users and two different road authorities responsible for traffic signal control are investigate by van Zuylen and Taale (2004) in an analytic, static setup, and by simulation. In a Stackelberg game, one of the road authorities takes the role of the leader. The other authority and the road users react to the decisions of the

[1] see `http://mctrans.ce.ufl.edu/featured/dynasmart/`, last access 17.05.2013

[2] The Cournot-Nash equilibrium is defined as follows: "In a noncooperative game between a traffic authority and highway users, a combination of strategies (g^*, h^*) is a Cournot(-Nash) equilibrium $\Leftrightarrow$ control plan g^* is the traffic authority's best response to flow h^*, and flow h^* is the users' best response to control plan g^*" (from Chen and Ben-Akiva, 1998).
The Stackelberg equilibrium improves over the Cournot equilibrium in that the traffic control authority anticipates the users' reactions: "..., a strategy combination (g^*, h^*) is a Stackelberg equilibrium $\Leftrightarrow$ it solves the following bilevel programming problem: $\min_g Z[g, h^*(g)]$ such that flow $h^*(g)$ is the users' best response", where Z is the traffic authority's objective function (after Chen and Ben-Akiva, 1998).
In the system optimum (= monopoly game), the traffic authority also controls the user flows; the optimization problem becomes: $\min_{g,h} Z[g, h]$.

leader. Results for the analytical solution differ according to the leader of the Stackelberg game. Same holds for the presented dynamic simulation studies. Best results, however, are reported when both authorities cooperate and anticipate the road users' reaction on the traffic signal control. Noteworthy, the simulation model takes into account physical queue length (Taale and Van Zuylen, 2003) and is refined in further works to capture spill-back effects between links of the transport network (Taale, 2008). In a more general context, the importance of modeling physical queues for dynamic assignment is emphasized by Daganzo (1998). The temporary limited effect of queue spill-back in conjunction with a dynamic assignment may lead to situations where the equilibrium is no longer unique.

Physical Queues

In general, agent- and mechanism-design both can benefit from use of game theory (Russel and Norvig, 2010, pp. 666). Most mentioned approaches use classical game theory and the Nash equilibrium assumptions, i.e., the impact of a strategy is assumed to be fully known by all players. The approaches modeling interactions between traffic control and travelers that take time-scales into account use iterative solution techniques. Traffic assignment is solved iteratively and at least partly via simulation. Hu and Mahmassani (1997) report amongst other as result the number of "days" [3] until convergence as measure of effectiveness. To describe the dynamic interactions between players, also evolutionary game theory can be used (e.g. Hofbauer and Sigmund, 1998). While classical game theory looks for equilibrium solutions, evolutionary game theory searches strategies that are evolutionarily stable (Bazzan, 2009, p. 348). Thus, use of evolutionary game theory helps when learning processes are included in the model. E.g., the agents that learn to control signalized junctions proposed in Bazzan (2005) are useful if they are able to learn quickly but useless if learning takes infinite time. Evolution seems plausible on both sides of the game – Bazzan et al. (e.g. 2008) signalized junctions *and* car drivers are modeled as learning agents.

Agents & Evolutionary Games

Illustrative Example

The following example considers two aspects of the mutual interaction between traffic signal control and travelers – the time scales of interaction and the physical representation of queues. The interaction of traffic signal control might be interpreted as Stackelberg game. In case of a fixed-time controlled network the transport authority is the leader setting up a signal control strategy while travelers adopt. This leader-follower relation is no longer clear if a fixed-time signal control is replaced by a traffic-responsive control. But first, some simulation results on a small, toy network are presented.

Interpretation as Game

[3] one iteration represents one day

5.1.2. Scenarios and Simulations

Base case

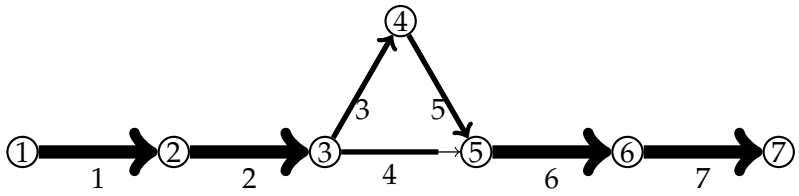


Figure 5.1.: Network. The widths of the links indicate their capacities, but are not proportional.

Links	Length $[m]$	Lanes	v_{fs} [m/s]	$c_{flow}\,[veh/h]$	$c_{storage}\,[veh]$	tt_{fs} [s]
1 & 2 & 6 & 7	1000.0	3	10.0	36000.0	1000	100.0
3 & 5	1000.0	1	10.0	7200.0	200	100.0
4	1000.0	1	10.0	**1800.0**	133	100.0

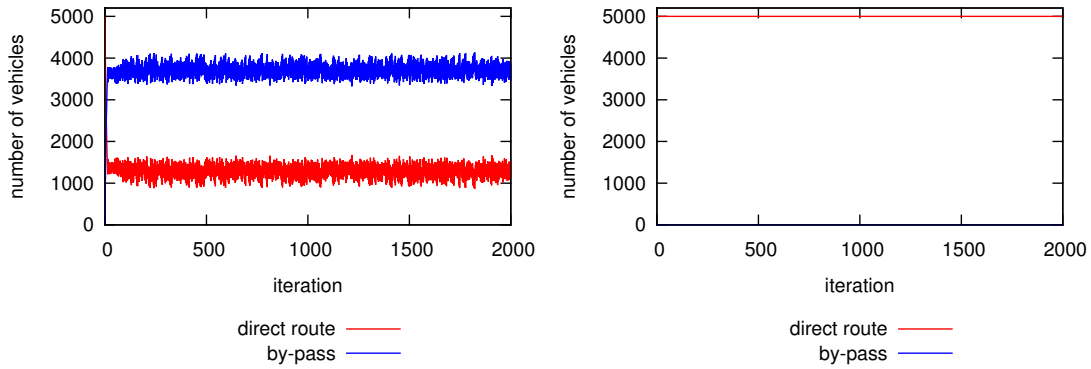
Table 5.1.: Network parameters

Network Let us start with the network of Fig. 5.1, described in more detail in Table 5.1. The important feature is that there is a bottleneck on link 4 shortly before node 5, and an alternative, but longer route to node 5 with higher capacity. The situation could be caused by link 4 going through the center of a city and links 3 & 5 providing a by-pass.

Demand Let us further assume a *demand* of 5000 vehicles (*veh*), originating on link 1, with a rate of 2 veh/sec (i.e., 7200 veh/h), with its destination on link 7. The demand is constructed in a way that

- the bottleneck link 4 by itself *does not* have sufficient capacity
- the bottleneck link 4 together with the alternative route *does* have sufficient capacity
- all upstream and downstream links, i.e., 1, 2, 6, and 7, always have more than sufficient capacity

Iterations Now let us assume a situation where that demand occurs repeatedly every day. Let us also make the usual assumption that travelers follow her or his own interest, i.e., that the system goes to a Nash equilibrium. The actual implementation is as follows:



(a) Base case (b) After expansion of the direct route

Figure 5.2.: Number of users of the direct route (link 4) and of the bypass (links 3 & 5)

1. Initially, every traveler has a plan that corresponds to the fastest free speed route, i.e., via link 4.
2. The traffic traffic flow simulation (Sec. 2.2 & 3.2.4) is executed. The travel time of every individual plan is memorized by the corresponding traveler.
3. In every iteration, 10% of all travelers obtain an additional plan, which contains the route that would have been approximately fastest in the previous iteration[4].

   All other travelers choose one of the existing plans(= route) according to the Random Utility Model (Sec. 2). The travel time experienced in the network loading is used as (negative) Utility V.
4. Goto 2.

The iterations are run until a stable outcome is clearly discernible, see, e.g., Fig. 5.2a.

For the synthetic scenario here, the travelers have no other alternative than taking the other route. In reality, they may also consider different departure times, different modes of transport, etc.

Results The result, as Fig. 5.2a shows, is that about 1100 travelers use the direct route while the remaining approximately 3900 travelers use the bypass. This is accompanied by congestion upstream of the bottleneck on link 4, see Fig. 5.3a, that builds up with the early travelers, and dissolves with the final ones, see Fig. 5.3b. Clearly, the congestion makes the direct route option appropriately slower, until the alternative becomes equally attractive.

[4] That route depends on the traveler's departure time. MATSim contains a time dependent router that aggregates link travel times into time bins of 15 mins. That temporal resolution is insufficient for the situation here. For this study the size of the time bins is set to 1 second.

(a) The queue is balanced (b) The queue resolves

Figure 5.3.: Base case, screenshots

The sum of all travel times is approx. 890 h, on average 641 sec per traveler.

Expansion of the direct route

One may consider the expansion of the direct route. The iterations were thus repeated with everything the same except that the flow capacity of link 4 was expanded to 5400 veh/h. Note that this is still not enough to serve all demand on the direct route under stationary conditions.

As a result, all travelers use the direct route (Fig. 5.2b). However, the sum of all travel times has *increased* to approx. 1161 h, or 836 sec on average per traveler. As Daganzo (1998) pointed out, this is another variant of the Braess paradoxon (Braess, 1968), where the construction of additional infrastructure may in fact lead to a *de*crease in performance. The reason here is that the congestion before the direct route on link 4 spills over node 3. That is, travelers are already caught in the direct route congestion on link 2.

The wait time in a queue, w, is given by the number of vehicles in the queue, n_{queue}, divided by the service rate, q_{out}:

$$w = n_{queue}/q_{out} \; .$$

The critical point where congestion spills over from link 4 to link 2 is when the queue length that is necessary to balance the travel times on the alternative, given by $w = tt^{alt}$, is equal to the storage capacity of the link:

$$c_{storage} \stackrel{!}{=} n_{queue} = tt^{alt} \cdot q_{out} \; .$$

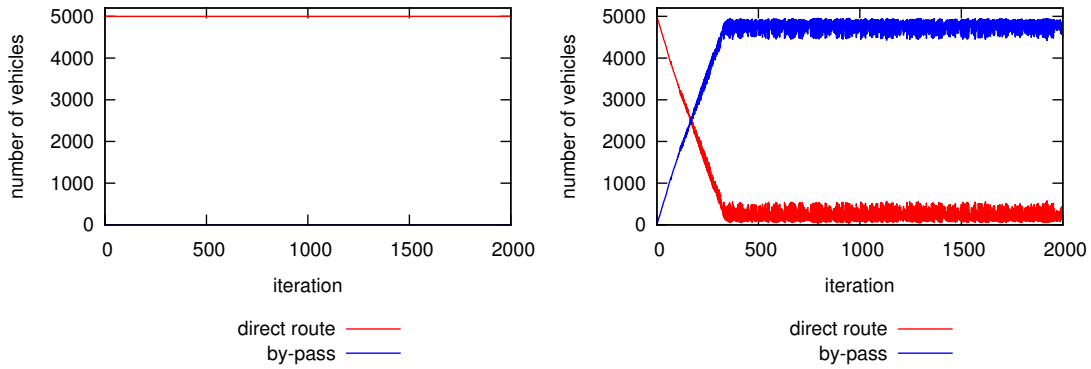
From this, one obtains the critical flow capacity of the link:

$$q_{out}^{crit} = c_{stor}/tt^{alt} \; .$$

Inserting $c_{storage} = 133.3\, veh$ and $tt^{3\&5} = 200\, sec$ one obtains

$$q_{out}^{crit} \approx 0.66\, veh/sec \approx 2400\, veh/h \; .$$

Any flow capacity *above* that number will lead to a balancing queue that is *longer* than link 4 and thus spills back into link 2. When a traveler in that queue reaches node 3,



(a) Expansion of direct route + adaptive signal (b) Expansion + adaptive signal + initial shock

Figure 5.4.: Number of users of the direct route (link 4) and of the bypass (links 3 & 5)

from there on link 4 is faster than the bypass. Therefore, under Nash conditions nobody selects the bypass anymore.

One thing to remember is that, under Nash conditions, "balancing" queues need to get longer when the capacity of the bottleneck is expanded. The queue only goes away when the capacity of the bottleneck is expanded sufficiently to serve *all* demand.

Adding an adaptive signal

Now, let us put an adaptive traffic light at node 5 which gives priority to link 5 over link 4[5]. This might be motivated by a transportation policy which attempts to make people use the bypass around the city instead of the direct route through the city.

The simulation (Fig. 5.4a) converges to the solution that uses the direct route, in spite of the adaptive signal that gives priority to the bypass. Curiously, however, it is possible to make the iterations converge to a solution that uses the bypass when there is an initial "shock" in the form of red light on link 4 for the initial 600 seconds of the simulation (Fig. 5.4b). Under the current setup the initial red on link 4 could also be achieved by additional vehicles that leave the bypass. The adaptive signal would then give right of way to this vehicles.

The existence of two stable states is consistent with the predictions of Daganzo (1998). The reason why the second state is also stable is that the initial shock makes the adaptive signal operate in a way that reduces capacity on link 4. As a result, the "balancing" queue gets short enough so that an equilibrium decision can be made at node 3. This,

[5]The traffic light switches link 4 to red light when a vehicle arrives at the end of link 5. If there are no vehicles at the end of link 5, link 4 gets green light.

in turn, generates enough traffic flow along the bypass to keep the adaptive signal in a low capacity state for link 4.

However, in contrast to what one might expect, the system does not reach spontaneously one or the other state depending on randomness during the transient phase. the initial shock is necessary to push the system into the other state. The reason lies in the interaction between the within-day dynamics of the system and the day-to-day dynamics of the evolutionary learning. For this, let us look at the vehicles in the sequence in which they enter the system:

- When the first vehicle arrives at node 3, taking the direct route is always the best option.
- The same holds for the 2nd, 3rd, 4th, ... vehicle. As a result, a queue forms upstream of the bottleneck.
- The interesting point is when that queue reaches node 3. Since the bottleneck is operating at capacity 5400 veh/h, we know from Sec. 5.1.2 that in this situation link 4 remains the faster option.

  In consequence, the vehicle that joins the queue when the queue has reached node 3 will select link 4.

  So will, for the same reason, all following vehicles: They may join the queue upstream of node 3, but once they have made it to node 3, the direct route is faster.

Overall, for all vehicles link 4 is the Nash solution.

The other state is not even stable when the system is started in that state. The problem is that for early vehicles, it is always better to take the direct route. However, as long as nobody takes the bypass, the capacity reduction for the direct route is not activated. And, as long as the capacity reduction for the direct route is not activated, the queue grows beyond node 3 while at node 3 the direct route remains faster.

Thus, the only way to stabilise the bypass solution is by an initial shock that keeps the adaptive traffic signal activated long enough until the first "normal" travelers have reached it via the bypass.

Replacing the adaptive by fixed-time signal control

Assume now the same situation as in Sec. 5.1.2, i.e., that the capacity of the direct route is expanded, and a traffic signal is added at node 5. In contrast to the adaptive sig-

nal from Sec. 5.1.2 now a fixed-time control is used. The fixed-time schedule gives a capacity of 1800 veh/h to link 4, and 7200 veh/h to link 5.[6]

It should immediately be clear that this does not need to be simulated, since it represents the same situation as described in Sec. 5.1.2. Thus, the bypass solution will be stable. Since the bypass solution also leads, in the average, to lower travel times, this means that *the fixed-time signal control is able to enforce a better solution in a situation where the adaptive signal control is unable to do so*.

5.1.3. Discussion

Expansion Storage vs Flow Capacity

Clearly, the results hinge at the assumption that the flow capacity expansions of link 4 are not accompanied by storage capacity expansions. More practically speaking, capacity expansions usually go along with more lanes, which automatically increases storage capacity. This assumption is, however, no longer fulfilled when the bottleneck is rather short, such as Fig. 1 implies. In such situations, it is quite plausible that flow capacity is expanded while storage capacity remains constant. Expansion of flow capacity without an increase of storage capacity are also plausible when a fixed-time signal control is replaced by a traffic-responsive control. Lämmer and Helbing (2010), for example, argue that responsive signals are able to use slack capacity that fixed-time signal control is not able to use. While we agree with this argument, it nevertheless needs to be balanced with the loss of influence over the drivers' routing as outlined above. It may also be possible to combine adaptive and fixed-time strategies to reap the combined benefits, for example by keeping the adaptive control in desired capacity range.

Importance of Queues

The presented results illustrate the importance of the spatial dimension of queues, that can also be found in (e.g. Daganzo, 1998; Burghout and Wahlstedt, 2007; Taale, 2008). In a model with point queues the situation where the bypass is not used at all would not occur at all (Daganzo, 1998). With point queues the link upstream to the bottleneck link (link 2 in Fig. 5.1) is not affected by the spill-back. That implies that at some point diverting to the bypass is less costly than waiting in front of the bottleneck.

Thus, traffic signal control approaches that do not take spill-back effects into account may not be able to predict such situations correctly. Some of the approaches mentioned in Sec. 5.1.1 are aware of the problem. TUC implements a "gating feature" (Diakaki et al., 2002) that prevents down-stream links from oversaturation. The same objective has the solution proposed by Lämmer and Treiber (2012) within the signal controlled sub-network upstream traffic signals are responsible to prevent oversaturation of downstream junctions. For the network in Fig. 5.1 an additional traffic signal at node 3 would be required that prevents link 4 from oversaturation.

[6] In the simulation, this can for example be achieved by expanding the direct route to 7200 veh/h and giving it 25% of the green split, and expanding the bypass to 9600 veh/h and giving it 75% of the green split. In practice, it could also be achieved by just adjusting the layout of the intersection.

Other approaches predict traffic states for a certain time horizon considering spill-back and oversaturation (e.g. Taale, 2008; Gartner and Stamatiadis, 2007). This may resolve the problem if time-scales for prediction and user reaction are modeled accordingly.

Game theory & Time-scales

The adaptive signal control used for this study is indeed too simple to be compared with the sophisticated approaches from literature. It is constructed as extreme traffic-responsive opposite to fixed-time control and can be used in conjunction with game theory to explain the observations. With fixed-time control, the demand follows the signals which is a Stackelberg game, where the fixed-time control is the strategy of the leader, e.g., a transport authority. In case of adaptive control, the time-scales of user reactions should be considered. Then, the game theoretic interpretation is no longer obvious (Taale, 2008, pp. 11). For the provided example, it is important to note that the fixed-time control, i.e., the Stackelberg game, can enforce a policy that would not be Nash-stable by itself. The adaptive control, however, cannot enforce the solution. Thus, can we improve the adaptive control to anticipate user reactions perfectly?

Braess NP-hard

Roughgarden (2006) shows that solutions for Braess network instances are NP-hard if travelers route selfishly. To determine a network design that optimizes routing for a Braess like network, one has to build the entire network. In the worst case, one has to decide link by link if it is better to use the link or not.

Following Daganzo (1998), the network used in this section is a special instance of a Braess instance. The overall question is no longer to use a link or not but rather how much maximum flow a link should provide. The latter includes the cases where flow is 0 or maximal. This is, the problem can be seen as a Braess instance with an even bigger solution space. One may assume that the bigger solution space is not reducing complexity of the problem. Thus, strategies for traffic signal optimization that take travelers' reactions into account have to solve a NP-hard problem.

Approximation

More practically speaking, NP-hard means that solution algorithms for the exact problem cannot be found with bound computation time. The classical solution technique is approximation. E.g. by using a rolling horizon strategy one can predict short term reactions on transport demand on actuated signal control by a dynamic traffic assignment model (Gartner and Stamatiadis, 2007). Due to the nature of rolling horizon there is a limitation of the predicted time-scale. If this time horizon is long enough to predict long-term user reactions cannot be guaranteed. Furthermore, in order to save computation time the forecasting model might be not accurate enough to predict all effects.

Simulation

This can be resolved by simulations that represent traffic and user behavior more realistically (Hu and Mahmassani, 1997). The complete approximation methodology for the combined traffic and assignment and control problem can be run in conjunction with a more elaborated simulation model (Burghout and Wahlstedt, 2007). No proof can given that the chosen simulation and scenario reveals all possible effects of a adaptive signal control. However, there is at least a possibility to detect them.

Micro-simulation based assessment of solutions for the combined traffic assignment and control problem also enables us to compare different methodologies facing the problem. Most of the more modern traffic-actuated signal control strategies reviewed in Sec. 5.1.1 are of at least partial algorithmic nature and cannot be evaluated by models that need a mathematical description of the signal control under consideration. Microsimulation, however, can asses both types.

5.1.4. Conclusion

CTAC & Simulation

This study looks at the combined traffic control and assignment problem from several perspectives. It investigates how the quality of the solution changes when one side learns faster or slower than the other. Learning more slowly makes one the leader in a Stackelberg game (e.g. Nagel et al., 2004). Faster learning on one side may also change the type of game completely. Furthermore, the section illustrates the importance to include spill-back in transport models.

Overall

Overall, we would argue that before a congested region switches from a fixed-time to a traffic-responsive signal control, it should evaluate the consequences with a model that includes demand reactions to changes in the signal control. Evaluation should take place with a distinct model for demand reactions than the model that is eventually used to predict effects of signal control.

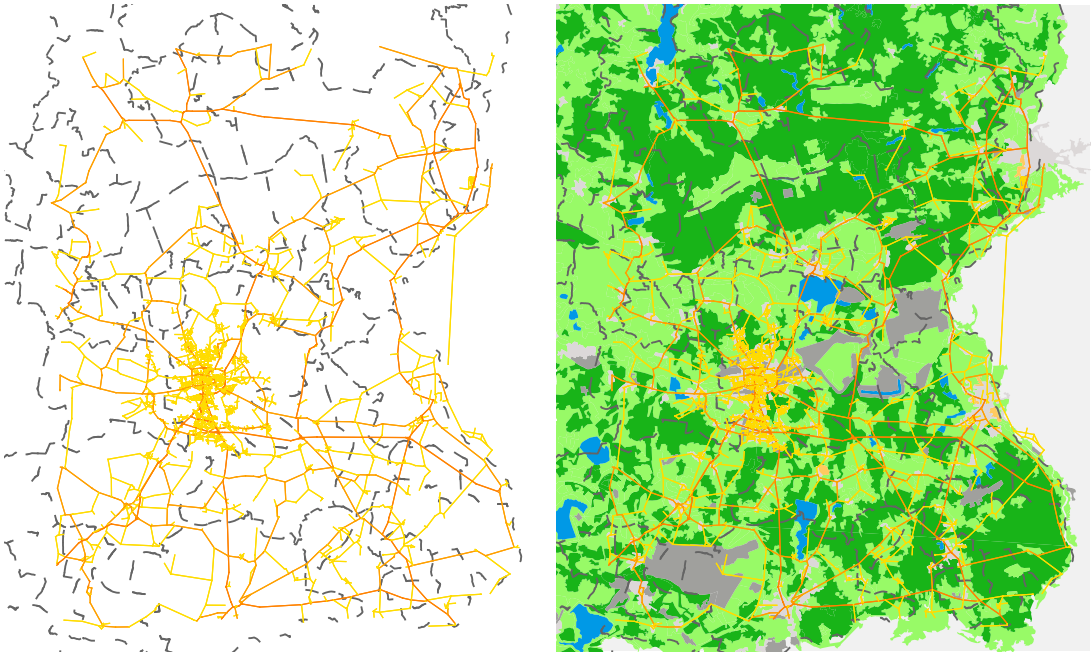
To our knowledge, very few simulation models do that. The reason, presumably, is that on the one hand one needs a dynamic model, since adaptive signal control unfolds over time. On the other hand, however, the (simulation) model needs to be able to treat regions large enough for re-routing decisions to make sense.

5.2. Cottbus Scenario

The network of the last section is clearly not realistic. To simulate a real-world scenario, at least some fixed-time signal schedules and junction layouts are required. Data for Zurich, Munich and Berlin was inspected. In part, fixed-time schedules are available. The matching of traffic signal locations to junction layouts was not covered by all data in a machine processable data format [7]. Within the ADVEST project, fixed-time control schedules and turn pocket layouts were recorded manually[8] for the work presented in Köhler and Strehler (2010). In the following a scenario is set up around this data.

[7] The Berlin network has approx. 2000 signalized junctions. Each is controlled by up to 4 fixed-time schedules for weekdays. On top of the fixed-time control some logic programs are set up for many junctions. The layout of junctions is given as technical drawing that is not connected to the control schedules. Converting this data manually is not an option.

[8] Acknowledgements to Martin Strehler, BTU Cottbus



(a) Cottbus network and municipality borders (b) Cottbus network and municipality borders

The scenario is located in the federal state of Brandenburg, in Germany. It covers the area of the administrative district "Spree-Neiße" that is enclosing the city of Cottbus, plus the City of Cottbus itself.

5.2.1. Network & Population

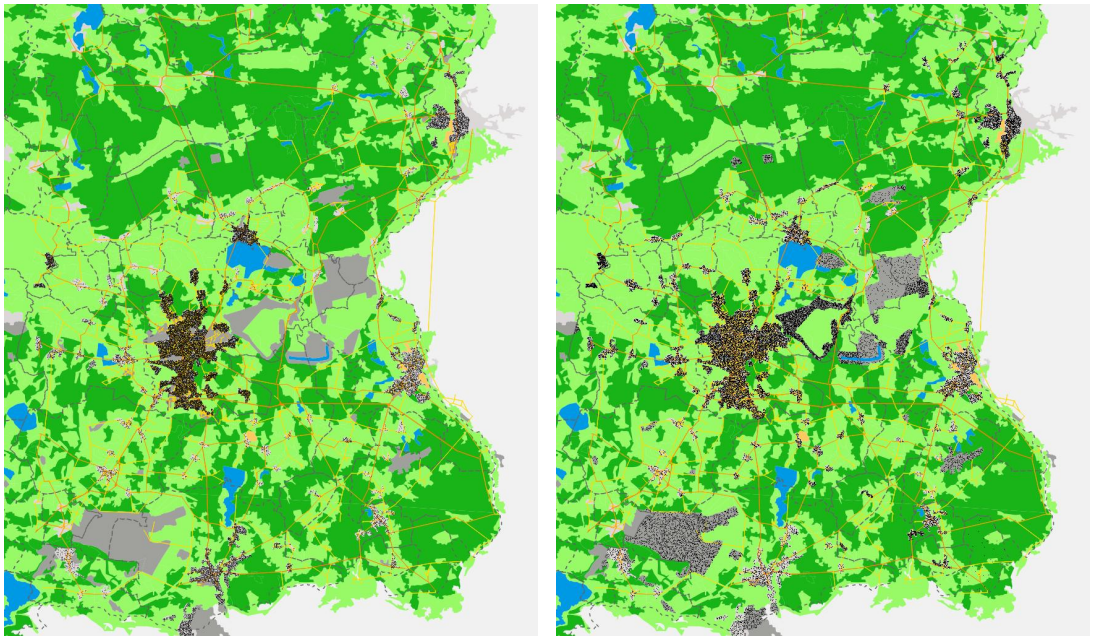
Network

The network is taken from openstreetmap data[9]. The network is created for an 100 % sample. In the administrative district "Spree-Neiße" only the main roads are included while for the city of Cottbus also side roads are considered. The network consists of 4'417 nodes and 10'600 links and is depicted in Fig. 5.5a. Fig. 5.5b shows the network on top of the "Corine Land Cover" landuse (European Environment Agency, 2011) provided by European Environmental Agency. In green forests and agricultural areas are depicted.

Population

In the city of Cottbus live around 100'000 inhabitants while approx. 128'000 people reside in the administrative district Spree-Neiße. The synthetic population used for the simulation is based on data taken from the German employment agency (Wiethölter et al., 2010). The data contains the number of commuters for each 2-tuple (home–work)

[9] state 09-2010, see `www.openstreetmap.org`, last access 03.10.2013. Technically, the open street map conversion of MATSim is used `OsmNetworkReader`.

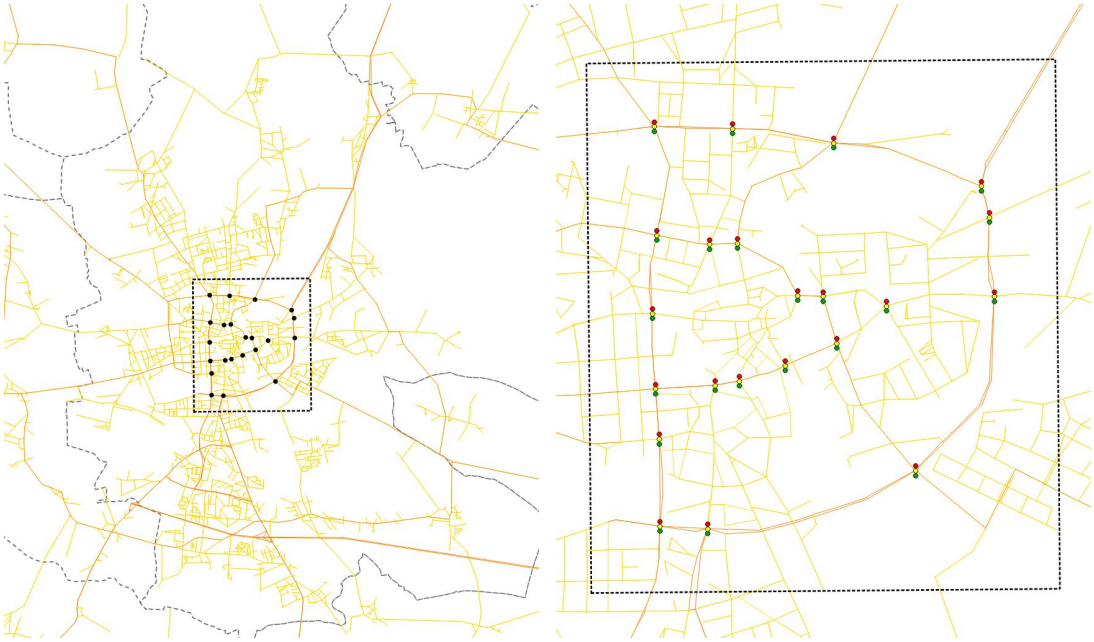


(c) Home locations (d) Work locations

Figure 5.5.: Synthetic population for the Cottbus scenario, geospatial locations of activities

of municipalities in Germany. Virtual persons in MATSim need a geographic coordinate for their activities. If this coordinate is drawn randomly, solely based on municipality boarders, home and work activity locations are uniformly distributed over all the area, i.e., most of them in woods and fields. Thus, activity locations are drawn randomly in combination with the landuse data. The coordinate has to be in the area of the municipality. In case of a home activity, it must be located in urban fabric areas while in case of a work location, also industrial or commercial areas are allowed. The resulting home activity locations are shown in Fig. 5.5c, while Fig. 5.5d shows activity locations for work.

The work activity must start between 7 and 9 am; initially every commuter starts at a random time in this interval and ends work 8.5 hours later. Work must end before 6 pm, afterwards no further utility is gained by performing an activity of type work. The typical durations of home and work activity are set to 15.5 hours and 8.5 hours, respectively. The modal split for the area of interest can be taken from the base year of ITP/BVU (2005) and is set to 55% car trips. This results in 33'479 commuters travelling by car.



(a) Location within city of Cottbus (b) Signalized area in detail

Figure 5.6.: Cottbus network, area with traffic signals

5.2.2. Traffic Signals

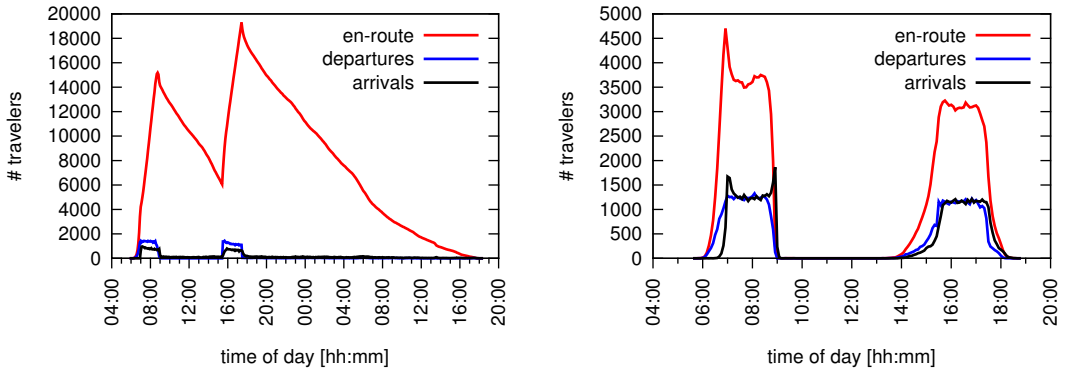
Fixed-time Control

The fixed-time control is taken from Köhler and Strehler (2010). Due to the higher resolution of the transport network, some of the originally recorded fixed-time control schedules are invalid and removed, data for 22 junctions is available. Fig. 5.6 shows their location on the transport network. All signal control plans have a cycle of 90 seconds and run all day (00:00 to 24:00). Green splits are taken from the system that was run in 2009, and offsets are the result of the optimization presented in Köhler and Strehler (2010). The demand used for optimization differs from the commuter demand used in this work. This reflects the typical situation of optimized fixed-time control: Signals are optimized to a certain demand once, but while the demand changes over time the fixed-time control is not re-adjusted (Bell and Bretherton, 1986).

5.2.3. Base Case

Simulation Setup

A *base case* for further Cottbus simulation runs is computed using the network, synthetic population and traffic signal control. The simulation is run with the commuter population until the outcome seems stable, in this case for 500 iterations. In each iteration, 10 % of the commuters can choose new routes while another 10 % can vary their departure times. The only available mode is car. Then innovation is switched off,



(a) 0th iteration

(b) 1000th iteration

Figure 5.7.: Cottbus base case: After 1000 iterations a route and time distribution is learned

i.e., another 500 iterations are run where no new plans are generated. Each commuter chooses out of a set of 4 plans using the multinomial logit model, see Sec. 2 for details. The network is created for a 100 % sample. The demand comprises commuter traffic, but other traffic is not included, e.g., commercial traffic. The scale parameters α_{flow} and $\alpha_{storage}$ of the traffic flow model are both set to 0.7.

Results

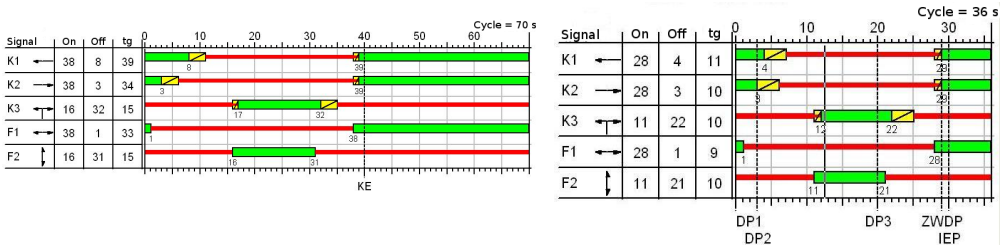
The resulting relaxed transport demand serves as input for the for further simulations. In contrast to the initial synthetic population the relaxed transport demand features a route and time distribution that is learned according to the constraints of the transport network and the utility function. The results of this learning process are shown in Fig. 5.7 that depicts the number of travelers departing, arriving or traveling over time of day for first and last iteration.

Discussion

Clearly, this scenario has many free parameters that are adjusted by rule of thumb only. For other available scenarios, more data and better calibration is available, but neither signal control data nor junction layouts. For a simulation of traffic signal control the Cottbus scenario is the best available scenario. In comparison to toy networks it features many artefacts of real-world networks.

5.3. Illustrative Application: Cottbus, Football Event

The Cottbus scenario is now used to illustrate the influence of traffic signal control. When it comes to a big event, the fixed-time control of the base case scenario is compared with a simple traffic-actuated control.



(a) Fixed-time schedule (b) Base schedule

Figure 5.8.: Schedules for traffic signal control (source: Titze (2008))

5.3.1. Traffic-Actuated Signal Control

Backgrounds

In this chapter a signal control strategy from industry called SYLVIA[10] is used (Schlothauer & Wauer Ingenieurgesellschaft für Straßenverkehr mbH & Co. KG, 2011, 2010). Originally developed by Siemens in the nineties (Krimmling, 1995), currently Schlothauer & Wauer[11], a German engineering company, is developing and supporting this signal control strategy. SYLVIA is not a closed, unique algorithm for signal control. Rather, it consists of several modules that can be used standalone or in conjunction.

Example

In this work, one of the main features of SYLVIA is applied, the traffic-actuated stage length control. This is based on pretimed fixed-time schedules. An example fixed-time schedule with a cycle of 70 seconds, three signals for car traffic (K1 – K3), and two signals for pedestrian traffic (F1, F2) is shown in Fig. 5.8a. For traffic-actuated stage length control, first the fixed-time schedule is compressed to a *base schedule* (Fig. 5.8b). Green of all signals is reduced to a minimal time, in this example to about 10 to 12 seconds. The cycle of the base plan is thus reduced to 38 seconds. Then, extension points are specified. An extension point is a point in the base plan where, if desired, the green time can be extended. A traffic engineer manually specifies extension points, minimal and maximal extension times for each signal, and a condition for extension. The condition for the green time extension of a signal is set with respect to the detectors that are available at the junction. The base schedule is then processed similar to a fixed-time schedule, i.e., a periodic timer controls when a signal is green, according to the green times specified in the base schedule. If the timer reaches an extension point, the condition for extension is checked. While the condition is true, the green time is extended and the timer is stopped. The signal switches to red, if the condition is false or a maximum extension time is reached. Then, the timer is reactivated and continues the processing of the base schedule. To pause and continue the timer during extension ensures, that the amount of all red time between conflicting signals is the same as for the fixed-time schedule.

Algorithm

In practice, the base schedules and conditions for extension are set up manually by a

[10] "SYstem Leipzig für die Verkehrabhängige Individuelle Steuerung von LichtsignalAnlagen"
[11] see `http://www.schlothauer.de/en/index.html`, last access 17.01.2011

specialist junction by junction. In this work, for sake of example, the base schedules and conditions are retrieved algorithmically from the fixed-time control schedules of the Cottbus base case. Phase ordering, amber times, and all red times are taken over. For all phases, the initial green time is set to 5 seconds. If after 4 seconds green time vehicles are still approaching the signals of a phase, the green time is extended up to a maximum, that is set to the corresponding phase length of the fixed-time schedule multiplied by 1.5, or at maximum to the length of the fixed-time schedule. Within one cycle, overall extension time is bounded by the difference between the cycle of the fixed-time schedule and the cycle of the base schedule. The bounded overall extension time implies that the later a phase is triggered by the base schedule, less overall extension time may be available. This is clearly not realistic in certain situations, but preserves the coordination of the fixed-time schedules.

5.3.2. Event: Football

The football stadium of Cottbus lies inside the ring road in the south-east of the city area and accomodates up to 22528 fans[12]. The local football club "FC Energie Cottbus" currently plays in Germany's second league so that it may happen that some kind of derby takes place on a normal weekday, thus interfering with the regular commuter traffic. In the area of the stadium around 2000 parking lots are available[13]. Thus in addition to the commuters, a synthetic population with up to 2000 persons travelling to the stadium by car is created. It is assumed that 25 % of these fans come from Cottbus, while the other 75 % come from the "Spree-Neiße" area, and that all fans start their trips between 17:00 and 18:00.

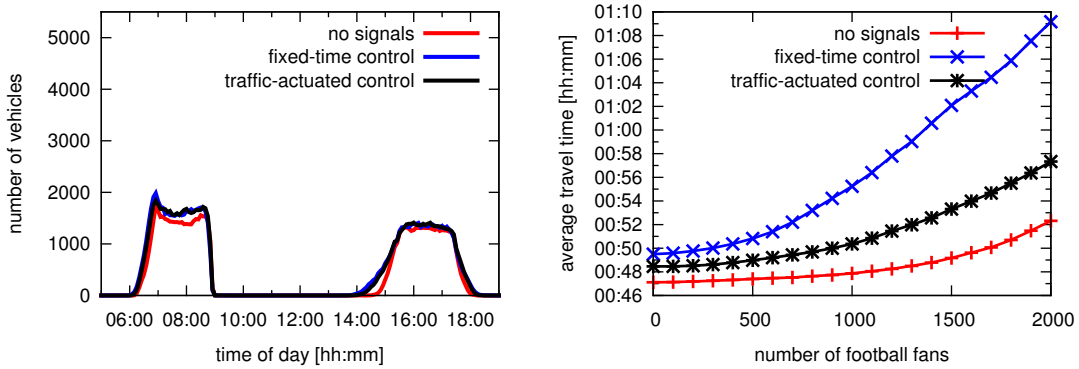
5.3.3. Run Sequences

The runs sequence of the base case is performed with three different signal control strategies: In a first simulation sequence, all traffic signals are switched off. This can be used as a lower bound for results concerning signal control since it assumes that vehicles are able to traverse a crossing without any accident, i.e., they are able to drive "through each other". The next sequence uses the fixed-time setup. In the third, final, sequence, all traffic signals are controlled by the traffic-actuated control.

5.3.4. Results

Simulation results for the commuter scenario are depicted in Fig. 5.9a. The number of vehicles simultaneously on the road is plotted over the time-of-day. The results are quite similar for all signal control strategies.

[12] `http://www.fcenergie.de/verein/stadion/home.php`, last access 14.02.2011
[13] `http://www.ssb-cottbus.de/sportstaetten/sdf/`, last access 14.02.2011



(a) No vs. fixed-time vs. traffic-actuated signal control, commuter traffic, iteration 1000

(b) Average travel time for unexpected event traffic, iteration 1000

Figure 5.9.: Simulation results

A change of signal control has more effect if some unexpected traffic occurs on the network. In the last iteration of the run sequences, in addition to the commuters 0 to 2000 vehicles drive to the football stadium of Cottbus during the evening peak. Fig. 5.9b plots the number of football fans on the x-axis, and the average travel time of all travelers on the y-axis. Without any additional vehicles, the traffic-actuated signal control leads to a gain of approx. 1 *min* per traveler. The more additional traffic is approaching the stadium, the more the traffic-actuated control saves travel time. In the case where 2000 additional vehicles are on the road, travel time savings reach ca. 15 *min* per traveler.

5.3.5. Computation Time

The computation was run on an Intel Xeon Westmere Hexacore architecture using 1 core for the microsimulation. Without simulation of lanes and traffic signals, one execution of the mobility simulation takes on average 13 seconds computation time. If the lanes are switched on, one execution of the mobility simulation takes 14 seconds. If additionally traffic signals are simulated it needs 16 seconds. Each iteration on average 17 seconds are required for scoring, replanning, and output. One complete run sequence (1000 iterations) takes 9 *h* and 12 *min*. The large number of iterations is necessary for a sufficient number of co-eolutionary learning iterations between the adaptive traffic signals and the adaptive agents.

5.3.6. Discussion

The presented example shows clearly the advantage of traffic-actuated signal control strategies. The traffic-actuated control used in this example, however, clearly benefits

from the fairly mild conditions of the base case. Fig. 5.9a shows, that switching on signal control has only little influence on overall traffic patterns. In the base case, traffic is only slightly jammed and a situation as described in Sec. 5.1 is unlikely to occur. The next section studies options for further calibration.

5.4. Optimization and Network Wide Analysis of Traffic Signal Control

5.4.1. Density, Speed, Flow

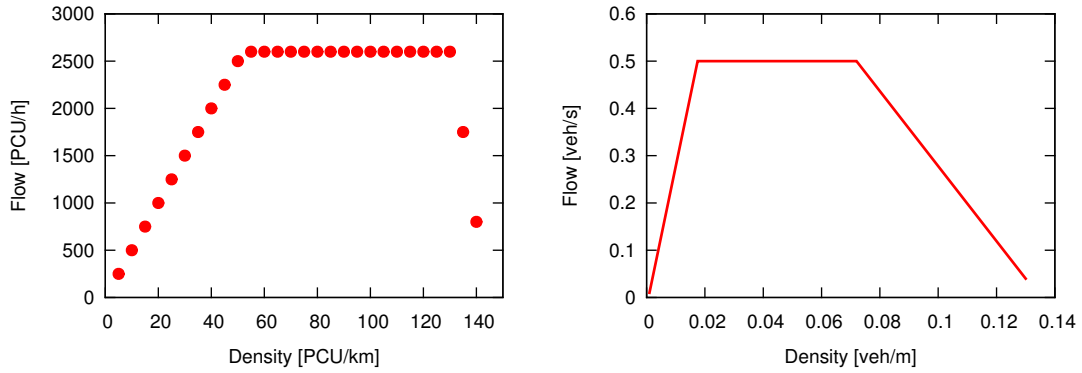
Backgrounds

Traffic density ρ, speed v and flow q on a single link compose the fundamental diagram. It is assumed that the general relation between the three variables is $q = \rho \times v$. Typically, three regimes exist on a single link – an under-saturated state, a saturated state and an oversaturated state. For the queue model used in this work, in the under-saturated state the number of vehicles traveling on the link is smaller than the maximum possible capacity of the link, while in the saturated stated the number of vehicles is fluctuating around this capacity. A single link cannot show the oversaturated regime. The oversaturated state arises, if vehicles cannot leave a link because of congestion on downstream links. The queue model produces travel times that correspond to these regimes (Simon et al., 1999). The oversaturated state is unrealistically narrow but captured (Agarwal et al., 2013). This is illustrated by the flow-density plot in Fig. 5.10a. Modifications to the queue dynamics can produce a more realistic oversaturated state (Charypar, 2008), Fig. 5.10b. MATSim currently comes with two mobility simulations that differ in the queue dynamics. The default mobility simulation applies the queue model proposed in Gawron (1998b); Simon et al. (1999); Cetin et al. (2003); Cetin (2005) as described in Sec. 3.2.2. The other is a reimplementation of the model by Charypar (2008) in Java (Waraich et al., 2009a).

Gartner and Wagner (2004) use the fundamental relation to analyse the influence of traffic signal coordination on the traffic flow of signalized arterials. Results are based on simulation studies that use the cellular automaton model by Nagel and Schreckenberg (1992) in a system with closed boundaries, the density is the variable that sets up the fundamental diagram. Using this setup, the delay that results from (lack of) signal coordination can be studied. It turns out, that coordination mostly affects the throughput of the arterial in the under- and oversaturated state. In the saturated state influence is less or not existing.

Macroscopic Relations

With the recent work of Daganzo (2007); Daganzo and Geroliminis (2008); Geroliminis and Daganzo (2008) an idea from the late 1960ties experienced a renaissance in transport research – the *Macroscopic Fundamental Diagram* (MFD). Similar to the fundamental diagram of single links, the MFD comprises the relation between density, speed and



(a) Queue model used in this work, source: Own figure that sketches the relation studied in Agarwal et al. (2013)

(b) Queue model with modified queue dynamics, source: Own figure that sketches the relation studied in Charypar (2008)

Figure 5.10.: Flow-density plots for different queue models for traffic flow

flow in a transport network. Empirical existence of the MFD can be shown based on detector and GPS data (Geroliminis and Daganzo, 2008).

Well-Defined?

Both, the fundamental diagram for single links as well as the MFD do not have to be well-defined for arbitrary networks. The selection of the subnetwork is important to retrieve a well-defined MFD from MATSim output (Zheng et al., 2012). Note, that the results of the paper apply the traffic flow model from Charypar (2008); Waraich et al. (2009a)[14]. Furthermore, in urban networks the location of measurements on road segments is important (Wagner et al., 2009). Summarizing, currently a lot of work is done to gather more knowledge about the fundamental relations of density, speed, and flow for transport networks. In this chapter, we will not join the overall discussion. The three values are used, to study the influence of different signal control strategies on several subnetworks.

Data Derivation

The spatial extend of the scenario is much bigger than the area covered by traffic signals. From the simulation output, we retrieve the time at that a vehicle enters or leaves a link. In an agent-based simulation, each individual vehicle can be tracked over its complete journey. Some constraints of detectors can be neglected. To capture effects of signalization for different spatial extends or subnetworks, all metering is based on links. This allows us to "zoom" to an area of interest. To stay consistent with other results for MATSim, derivation of density and flow is taken similar to Zheng et al. (2012):

- Space mean density in the subnetwork for time interval $\tau = [\tau_0, \tau_1]$ (referred as

[14] Information not obvious by reading the paper, but confirmed by personal communication with one of the authors.

"density" in the remainder of this chapter)

$$K_\tau = \frac{N_\tau}{L}; \quad \text{with } L = \sum_i l_i \cdot n_i\,,\ N_\tau = \sum_{k=0}^{\tau_1} e_k - q_k\,,$$

whereby for each link i of the subnetwork, l_i denotes the length and n_i the number of lanes, N_τ denotes the number of vehicles in the subnetwork during τ, and e_k and q_k the number of vehicles that enter or leave the subnetwork at time k, respectively.

- Space mean flow in the subnetwork for time interval $\tau = [\tau_0, \tau_1]$ (referred as "flow" in the remainder of this chapter)

$$Q_\tau = \frac{\sum_i (q_{i,\tau} \cdot l_i \cdot n_i)}{\sum_i (l_i \cdot n_i)}\,,$$

whereby l_i, and n_i denote the same as before, and $q_{i,\tau}$ is the number of vehicles that leave a link i of the subnetwork within τ.

In addition, travel time, speed, vehicle kilometers, and delay are metered. Except the speed, they are not required for the results in this section, but for the subsequent sections. Definitions are introduced here as they refer to the same metering.

For each vehicle j that enters or leaves a link i of length l_i in the (sub-)network under consideration, the following values are metered:

- Overall travel time (referred as "travel time")

$$tt = \sum_i \sum_j tt_{ij}\,,$$

whereby tt_{ij} is the travel time of vehicle j required to traverse link i.

- Space mean speed (referred as "speed" or v)

$$v = \frac{s}{t} = \frac{\sum_i \sum_j l_{ij}}{\sum_i \sum_j tt_{ij}} = \frac{\sum_i \sum_j l_{ij}}{\sum_i \sum_j \frac{l_{ij}}{v_{ij}}}\,;$$

the spatially weighted harmonic mean of all speeds metered per link i and vehicle j. If speed is given for a time interval, the time at that the vehicle has left the link specifies the interval.

- Vehicle kilometers (referred as "veh km")

$$veh\,km = \sum_i \sum_j l_{ij}\,;$$

the total distance traveled in the subnetwork.

- Overall delay (referred as "delay")

$$delay = \sum_i \sum_j tt_{ij} - tt_{fs,i};$$

the sum of all delay, whereby $tt_{fs,i}$ is the travel time required to traverse link i in freeflow conditions.

Base Case

First, we analyse the base case. The time interval for the measurements is set to $\tau = 5\,min$. Fig. 5.11a shows the flow-density relation for the full network. Note, that the scales of axis are normalized for each single study presented in the following to ease the comparison of the graphics. Density and flow shown in Fig. 5.11a are at low values. The full network for the urban and rural area might be too diverse. Thus, we focus on a subnetwork in the signalized area. Fig. 5.14a depicts the subnetwork in detail, that is referred as *optimization graph* in in the following. To analyse effects on traffic signal control, this area might be more informative. The flow density plot is depicted in Fig. 5.11b. The scatter can be explained by Fig. 5.11c depicting the number of travelers over time of day in the optimization graph subnetwork, each hour of day is drawn in a different color. The same coloring is used in the flow-density plot in Fig. 5.11d that shows only values metered during the morning peak. One can clearly observe that flow measurements are less when the network fills with vehicles, while at same density levels, measurements are higher when the network gets empty again. This might be an issue of the metering as vehicles are considered to be within the network as soon they enter the first link of the network. In contrast, flow is metered each time a vehicle leaves a link. Furthermore, the figure can explain the high peak at the beginning of the morning peak. This seems to be an artefact of the queue model used for simulation. While the network is filled with vehicles, queues get longer but are spatially not long enough to influence other links. Then, some links are jammed and the spill-back starts to affect other links upstream. The system then falls back to the state colored in green and blue in Fig. 5.11d.

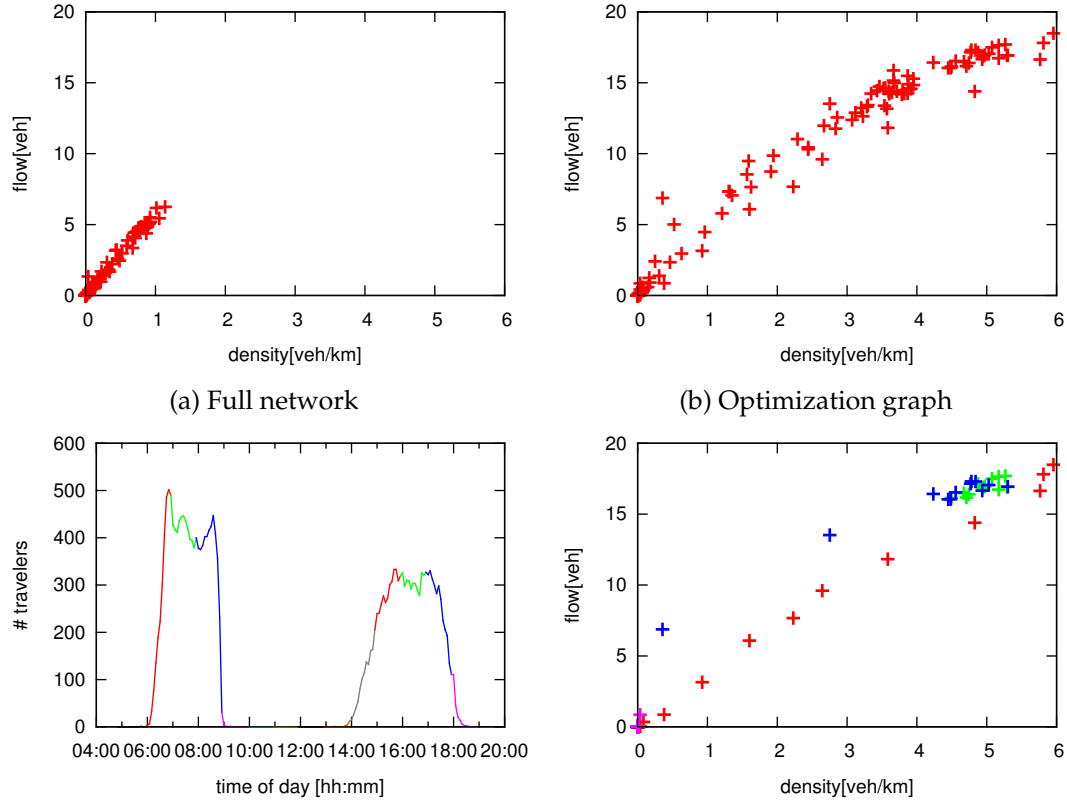
Base Case, Calibration

In terms of the model parameters α_{flow} and $\alpha_{storage}$ (Sec. 3.2.2), the base case is uncalibrated. So far, the two parameters were set by rule of thumb similar to many other studies that do not work with a full (100 %) demand. The flow-density plots might reveal further insights. The parameter α_{flow} is varied in $\{0.7, 0.5, 0.3\}$. The simulation is run with different choice dimensions for travelers. For the last simulation run $\alpha_{storage}$ is adjusted according to α_{flow}. All results refer to the optimization graph subnetwork.

Results

First, we look at the simulation in that travelers have no choice dimension. Therefore, the last iteration of the base case is rerun. Fig. 5.12a shows the flow-density diagram, Fig. 5.12b the speed-density plot. With higher values for α_{flow}, flow and speed are increased at lower density levels.

Then, route choice is added. The last iteration of the base case is continued for another 1000 iterations with the same setup as the base case. Route choice, however, is the only available choice dimension. Fig. 5.12c and Fig. 5.12d show the resulting flow-density



(a) Full network

(b) Optimization graph

(c) Optimization graph

(d) Optimization graph, morning peak

Figure 5.11.: Flow-density relations and travel patterns

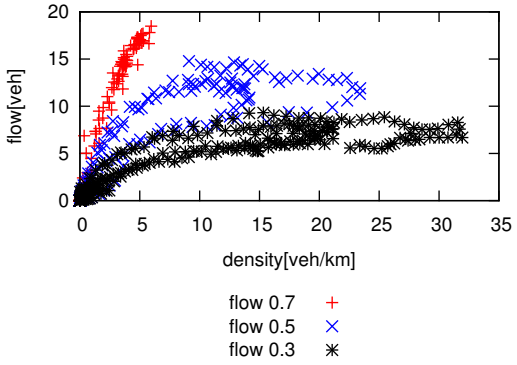
and speed-density plots, respectively. Densities are lower than in the no reaction simulation runs. Most travelers can avoid congested areas by the choice of new routes.

Results of simulations for route and time choice are shown in Fig. 5.12e and Fig. 5.12f. Densities decrease for all three values of α_{flow}.

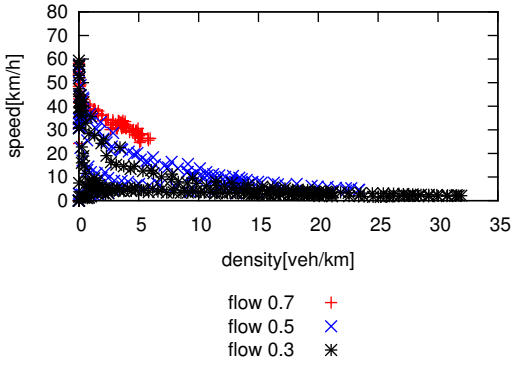
The joint variation of α_{flow} and $\alpha_{storage}$ is then depicted by Fig. 5.12g and Fig. 5.12h. Both plots have a similar shape as the previous results.

Findings

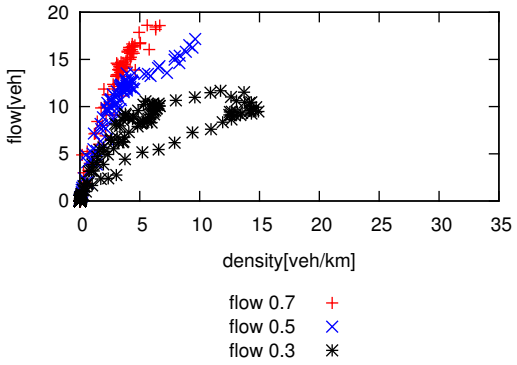
The results illustrate, how the queue model reproduces the general relation of $q = \rho \times v$. For the results with $\alpha_{flow} = 0.3$ at several density levels small decreasing tails of flow can be observed. The optimization graph subnetwork might be too diverse to show a well-defined MFD. The more choice dimensions are available, the more density stabilizes at a certain level. Given that traffic congestion is quite noticeable in Cottbus at certain times of the day, the model could be calibrated with $\alpha_{flow} = \alpha_{storage} = 0.3$. As discussions with local experts revealed, the subnetwork under consideration is only slightly jammed. Thus, the parameter settings $\alpha_{flow} = \alpha_{storage} = 0.7$ deliver plausible results and are not changed for the studies in the next section.



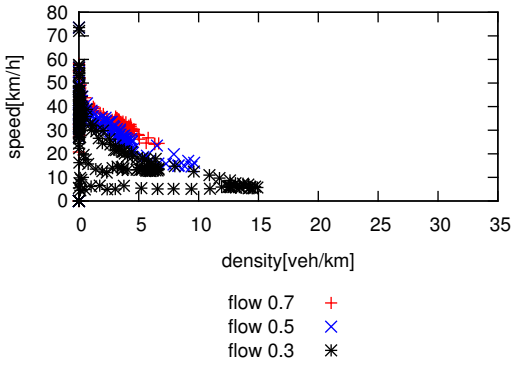
(a) No reaction



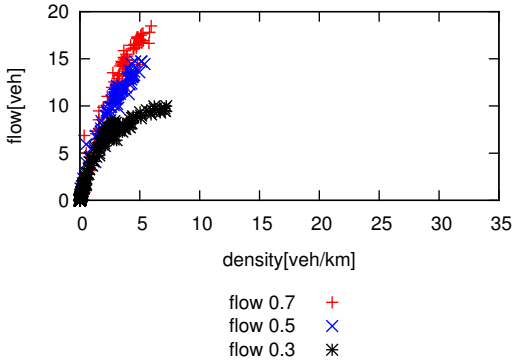
(b) No reaction



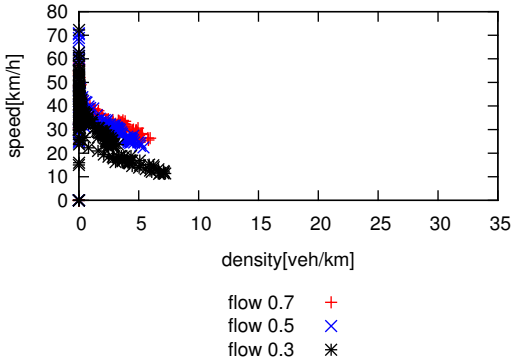
(c) Routes



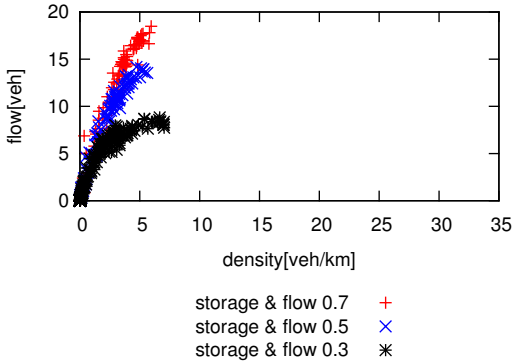
(d) Routes



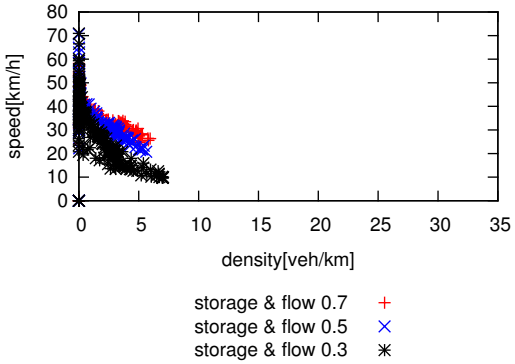
(e) Times & Routes



(f) Times & Routes



(g) Times & Routes



(h) Times & Routes

Figure 5.12.: Variation of system wide flow & storage capacity

5.4.2. Optimization Model for Fixed-time Control

Model

The optimization model, developed and run by Köhler and Strehler (2010, 2011); Strehler (2012) (BTU Cottbus), delivers a solution where signals are optimized jointly with the traffic assignment. At time of writing, the model is capable to optimize *coordination* of fixed-time signals and assignment. The inclusion of green splits in the optimization is planned for future work. For optimization, strict mathematical mixed integer linear programming (MIP) is applied. The model is based on a cyclically time-expanded graph $G = (V, E)$. Edges need some costs, that are the transit times at freespeed $t : E \to \mathbb{N}_0$ and capacities $u : E \to \mathbb{N}_0$. Origin-Destination (O-D) pairs with a certain amount of flow serve as input for traffic demand. These commodities start and end at vertices of the graph. Then, a static assignment is performed on the time expanded graph. No flow can be assigned to edges that connect vertices on that traffic is stopped by a red traffic signal. The objective function minimizes the sum of all travel times on edges times the flow assigned to the edge.

Complexity

Calculation of exact signal coordination is a complex problem. The problem solved by the optimization model is considered to be *NP-hard*, a proof is given by Köhler and Strehler (2010); Strehler (2012). In practical terms, NP-hard means that the time required for the computation of the solution increases exponentially with the size of the input[15]. Köhler and Strehler (2010); Strehler (2012) apply strict mathematical programming to cope with complexity. At time of writing, the model cannot be applied to large-scale networks and a huge number of commodities.

Previous Results

So far, the optimization model is evaluated with several scenarios (Köhler and Strehler, 2010, 2011; Strehler, 2012) that are all relatively small. Amongst others, also the traffic signals in the inner city area of Cottbus are optimized. The network used in these studies is shown in Fig. 5.13. The transport demand is created from local expert knowledge. Commodities between 10 different origins and destinations are modeled. The output of the joint optimization is then analysed with two simulation tools (ptv VISSIM and MATSim) and compared against "random" coordinations. The routes within the simulations are fixed to those computed by the optimization. Köhler and Strehler (2011) report that travel times vary only slightly between the two simulation models. Overall travel time is reduced by 24 %.

In a potential real-world application, however, travelers can not be forced to the routes provided by the optimization. But, by an improved coordination, there might be an incentive to travel along these routes. Further reactions could evolve, including departure time choice, mode choice, destination choice, etc. A first attempt to study the reactions on an improved coordination is presented in the following. The Cottbus base case of Sec. 5.2 serves as scenario. This requires a process to convert the scenario to the

[15] This is a rather coarse explanation of NP-hard problems, in theory there might be an algorithm solving the problem in polynomial time, that was not found, yet.

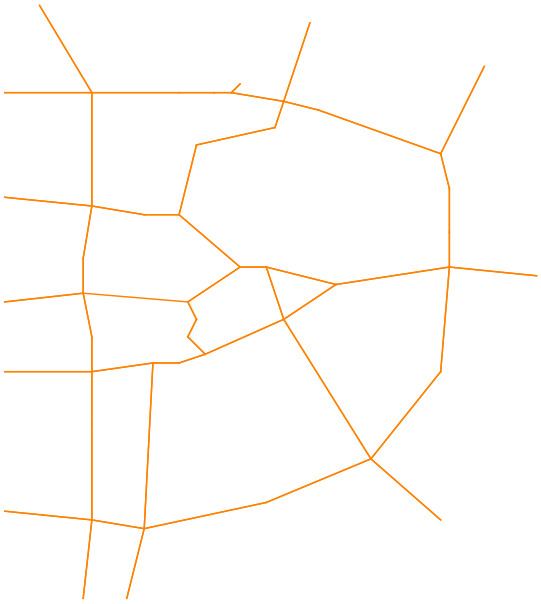


Figure 5.13.: Cottbus network, used for optimization of traffic signal coordination in Köhler and Strehler (2010, 2011); Strehler (2012) (source: own figure, network data provided by M. Strehler)

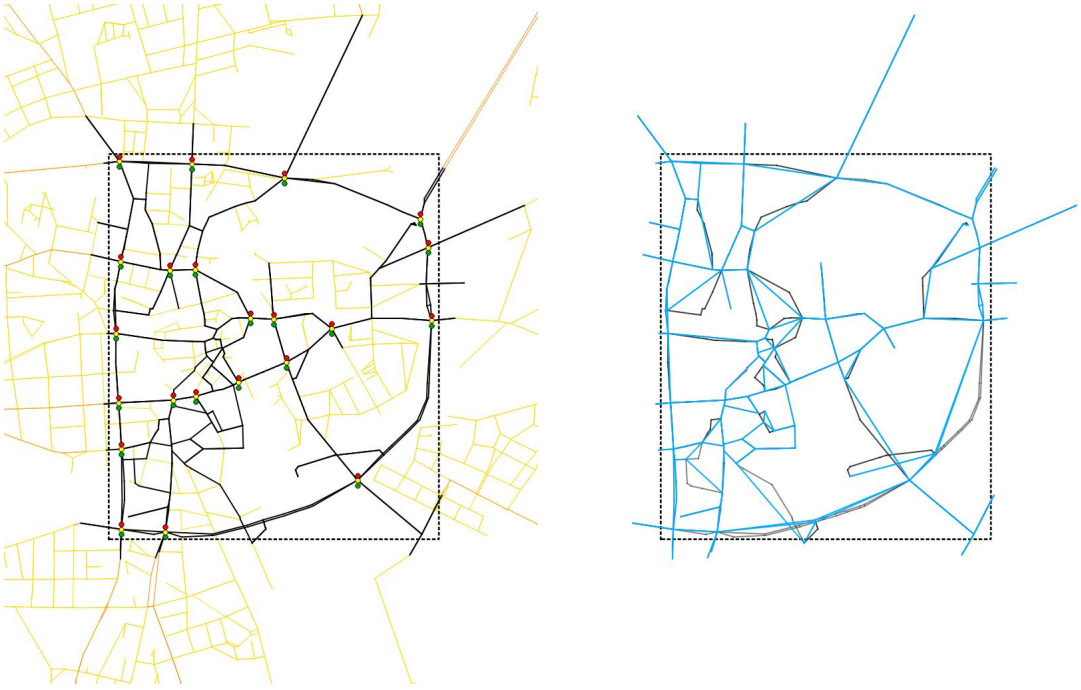
optimization model. Further, the size of the real-world instance has to be reduced until the problem is solvable by the optimization.

5.4.3. Conversion of Models

Overall Process

The conversion of fixed-time traffic signal control from the simulation to the optimization model is straightforward and is successfully used for the studies in Köhler and Strehler (2010, 2011); Strehler (2012). In contrast, converting the network and the demand of the base case from Sec. 5.2 to the optimization model is more complex. The semantics of input is different and the problem size has to be reduced. The following steps are described in the subsequent paragraphs:

- Reduce the network size so that it is computationally feasible for the optimization model.
- Convert the demand from x,y coordinates (MATSim convention) to aggregated commodities starting at nodes of the reduced network (optimization model convention).
- Adjust the parameters of the network and demand conversion so that the optimization model generates useful results within acceptable running times.
- Combine the single steps to a meaningful conversion process.



(a) Subnetwork (black) for the signalized area (b) Subnetwork as graph, not required vertices are removed

Figure 5.14.: Reduction of network size

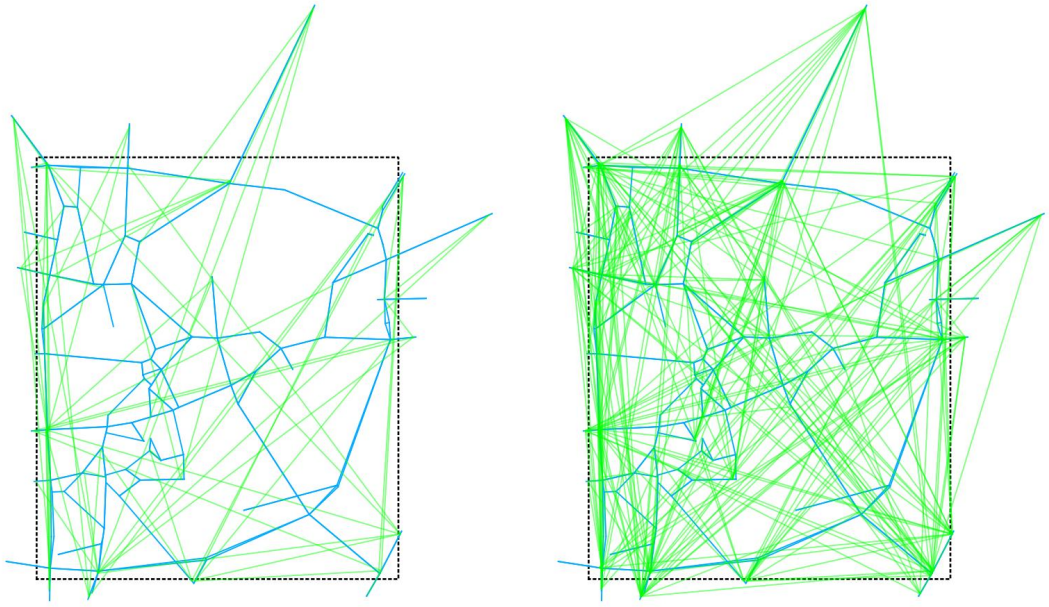
Reduction of Network Size

The bounding box around the signalized area is used to reduce the size of the network. All links of the full network are retrieved that start or end in this bounding box. This part or the network is still too big to be used as input for the optimization model. Its size is further reduced heuristically. Only links are selected that cover the bounding box and that have one of the following properties: Network Size

- A signal is located at one of the nodes that define the start or end point of the link.
- The speed limit of the link is greater than $10\,m/s$.
- The link is part of the shortest path between signalized nodes. For the shortest path calculations the length of each link defines the cost function.

The resulting subnetwork is shown by the black lines in Fig. 5.14a, the black dashed lines depict the bounding box.

This subnetwork still contains nodes that are not essential for a graph representation in terms of Sec. 3.1. Nodes with only one incoming and outgoing link in each direction that possess exactly same properties (vertices of degree 2 in an undirected graph) are Network to Graph



(a) O-D pairs with minimum flow of 50 vehicles

(b) O-D pairs with minimum flow of 10 vehicles

Figure 5.15.: Synthetic population of the Cottbus scenario converted to commodities

removed. The resulting graph is shown in blue in Fig. 5.14b. The gray network in the background is identical to the network shown in black in Fig. 5.14a. In a last step, crossings are unfold in the sense of Köhler and Strehler (2010, 2011); Strehler (2012). The resulting graph serves as input for the optimization model.

Conversion of Demand

Persons to Commodities

With the results of the base case presented in Sec. 5.2 a dynamic traffic assignment for the virtual population is available. This assignment is converted to static commodities under the assumption that 1 vehicle is equivalent to 1 unit of $flow$. The morning and the evening peak are converted separately, but using the same procedure. If a route of a virtual person traverses links of the subnetwork shown in black in Fig. 5.14a an O-D pair is created. The first link on the route that is contained by the subnetwork is the origin. The destination is the last link on the route within the subnetwork. If the O-D pair for the link was already created for a different route, the flow for the existing O-D pair is increased by one. If the subnetwork is traversed several times by a route, more than one O-D pair is affected. The resulting O-D pairs are then mapped to the

reduced graph representation shown in Fig. 5.14b and serve as commodities for the optimization.

Parameter Adjustments

Commodity Reduction

Flow is one for many commodities. Others have a flow of up to approx. 300 vehicles. After a first experiment, it turned out, that the optimization is not capable to calculate a good solution for all commodities. To limit the number of commodities, a threshold is introduced. If the flow of a commodity is less than the threshold, the commodity is refused. Fig. 5.15 shows the effect of this threshold: In Fig. 5.15a only commodities are depicted that have a minimum flow of 50. Fig. 5.15b shows O-D pairs with minimum flow of 10. The threshold of 50 removes around 55 % of overall flow in the subnetwork, the threshold of 10 approx. 28 %.

Network

The graph used for optimization needs transit times and capacities as attributes for each edge. As transit time the travel time at freespeed is taken from the simulation network. The commodities are gathered for a certain time interval ΔT_{OD} from the simulation, e.g., the duration of the morning peak. The simulation network specifies the maximum flow c_{flow} per Δt. These intervals can be used to resolve time dependency of capacity. To derive capacities of the static graph, also the effect of the refused commodities should be considered. If the threshold refuses γ of the overall flow in the subnetwork, the capacity of each edge in the optimization model is set to

$$(1 - \gamma) \cdot c_{flow} \cdot \frac{\Delta T_{OD}}{\Delta t}.$$

Note, that the subsequent results are computed with $\gamma = 0.0$ instead of approx. $\gamma = 0.55$ (minimum flow 50) or $\gamma = 0.28$ (minimum flow 10).

The traffic assignment that results from the optimization model for all commodities with flow greater 10.0 is shown in Fig. 5.16.

Conversion Process

The traffic assignment of the base case is converted for the morning (05:30 to 09:30) and the evening (13:30 to 18:30) peak. The outputs of the conversion are solved by the optimization separately. Then, only the coordination of traffic signals is converted back to the simulation model. In contrast to the base case, coordination for the morning and evening peak differs. The fixed-time control of the simulation exchanges the coordinations at noon.

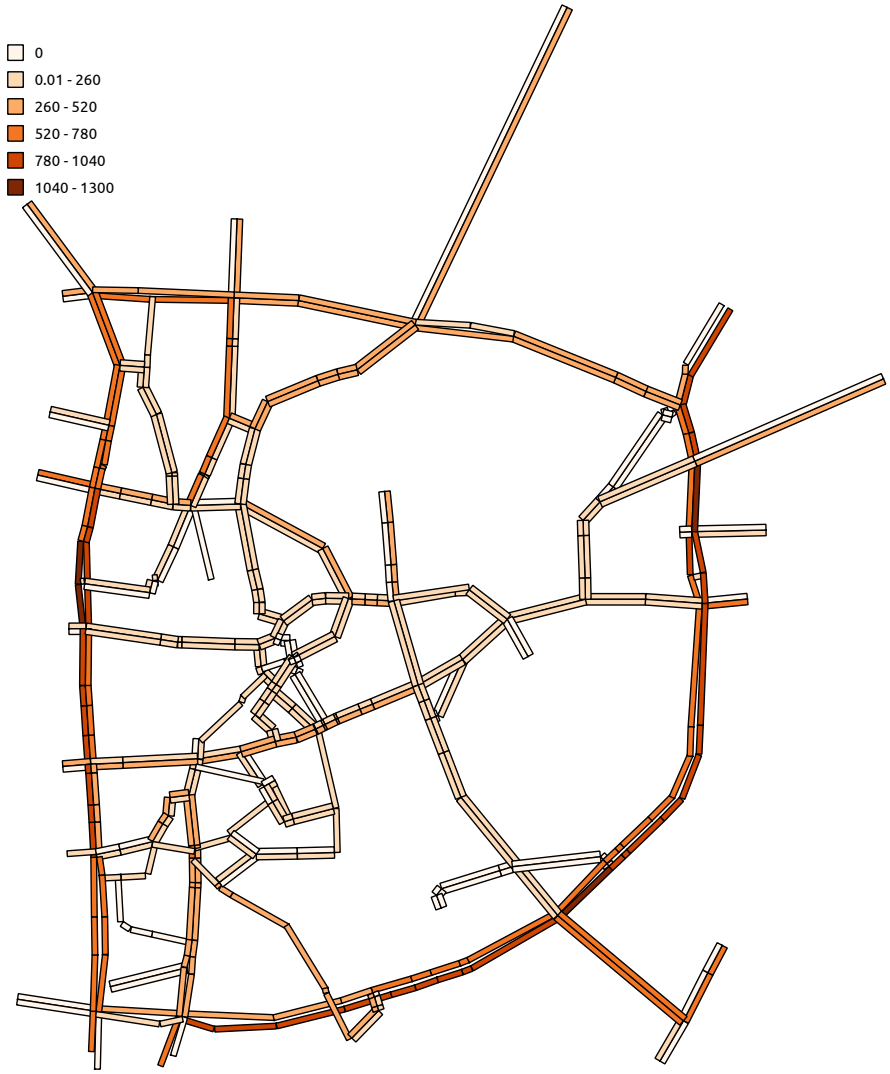


Figure 5.16.: Traffic assignment of the optimization model. Edges are colored by total flow assigned. (Source: Own figure, based on results provided by M. Strehler)

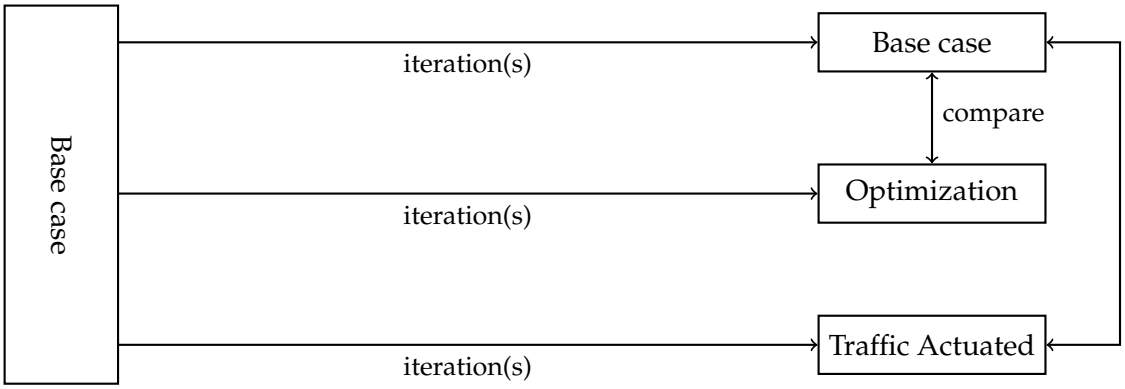


Figure 5.17.: Run sequence for simulation of different traffic signal control strategies

5.4.4. Simulation Setup

All simulation runs presented in the following use the outcome of the base case for the Cottbus scenario presented in Sec. 5.2. Thus, the traffic system is brought into equilibrium according to the choice dimensions "routes" and "times" with the baseline fixed-time control. Base Case

Four different setups of traffic signal control are simulated and referred as: Traffic Signal Control

- Base case: the fixed-time control of the base case.
- Optimization, commodities ≥ 50: The optimization of traffic signal coordination for the commodity ≥ 50 threshold. The solver was stopped with a duality gap of approx. 20 %.
- Optimization, commodities ≥ 10: The optimization of traffic signal coordination for the commodity ≥ 10 threshold. The solver was stopped with a duality gap of approx. 27 %.
- Traffic-actuated control: The traffic-actuated signal control of the illustrative application in Sec. 5.3.

Both optimizations were run[16] for at least 24 h, for the commodity ≥ 50 threshold, approx. 6 million MIP-Iterations were calculated. Due to the larger problem size, the solution for the commodity ≥ 10 threshold was run for approx. 2 million MIP-Iterations. The so-called duality gap serves as indicator for the quality of the calculated solution. There is, however, no valid interpretation for a reduction of the duality gap by 7 %. One question is if the improved gap results in significant improvements of the solution.

The system reaction is studied for two choice dimensions. In the first setup, travelers Choice

[16] Optimizations are run by Martin Strehler at BTU Cottbus, who kindly provided this details.

cannot change their routes or departure times. Then, travelers may change their routes while their departure times stay fixed.

Iterations

All simulation runs start with the outcome of the base case in iteration 1000. If travelers have no choice one iteration is sufficient to determine the outcome of simulation[17]. If route choice is enabled, the simulation setup of the base case is repeated without time choice. Up to iteration 1500, in each iteration 10 % of the travelers may change their routes. From iteration 1500 up to iteration 2000 only the logit model is active to switch between existing plans.

Interpretation

The setup of simulation runs is sketched in Fig. 5.17 and can be interpreted as follows: Traffic signal control is subject to change in a transport system at its current state. The traffic signal control was optimized by Köhler and Strehler (2010), but the demand has changed. The continued base case serves as reference if nothing is undertaken. The two optimizations and the traffic-actuated signal control are potential options for improvements. The effects of the different signal control strategies are compared to the continued base case.

5.4.5. Spatial Dimensions

The results of the simulation runs are analyzed for three (sub-)networks[18]. In the following, this three networks are referred as:

- Full network: The full network shown in Fig. 5.5a. Overall effects on the transport system can be observed.
- Optimization graph: The subnetwork that is used to construct the graph for the optimization model, i.e., the black colored network in Fig. 5.14a.
- Cottbus city area: The complement network of the optimization graph network with a limited extent, i.e., the black colored links in the dashed bounding box in Fig. 5.18.

5.4.6. Results

User Reactions

First, the outcome of the optimization, commodities ≥ 10 simulation run with route choice, is compared with the base case. An impression of user reactions on the change of traffic signal control is shown in Fig. 5.19a. Each dot denotes the location of an activity performed by a virtual person. Red dots mark activity locations of persons traveling through the signalized area if signals are not changed, but avoid the area in case signal

[17] This assumption holds as long as traffic signals are not learning over iterations

[18] The spatial resolution of the simulation output is based on networks. Input can be specified by geospatial coordinates. Some tests revealed that the complexity of code required for analysis of simulation results is increased if a truly geospatial resolution is used for analysis.

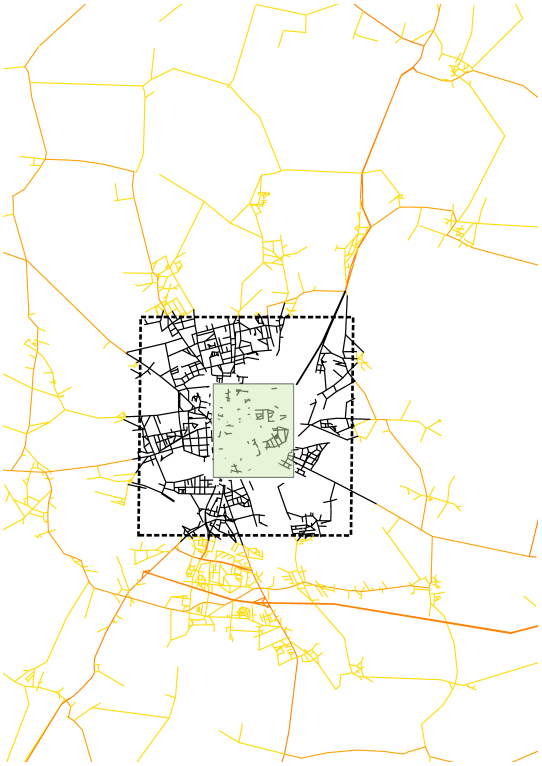
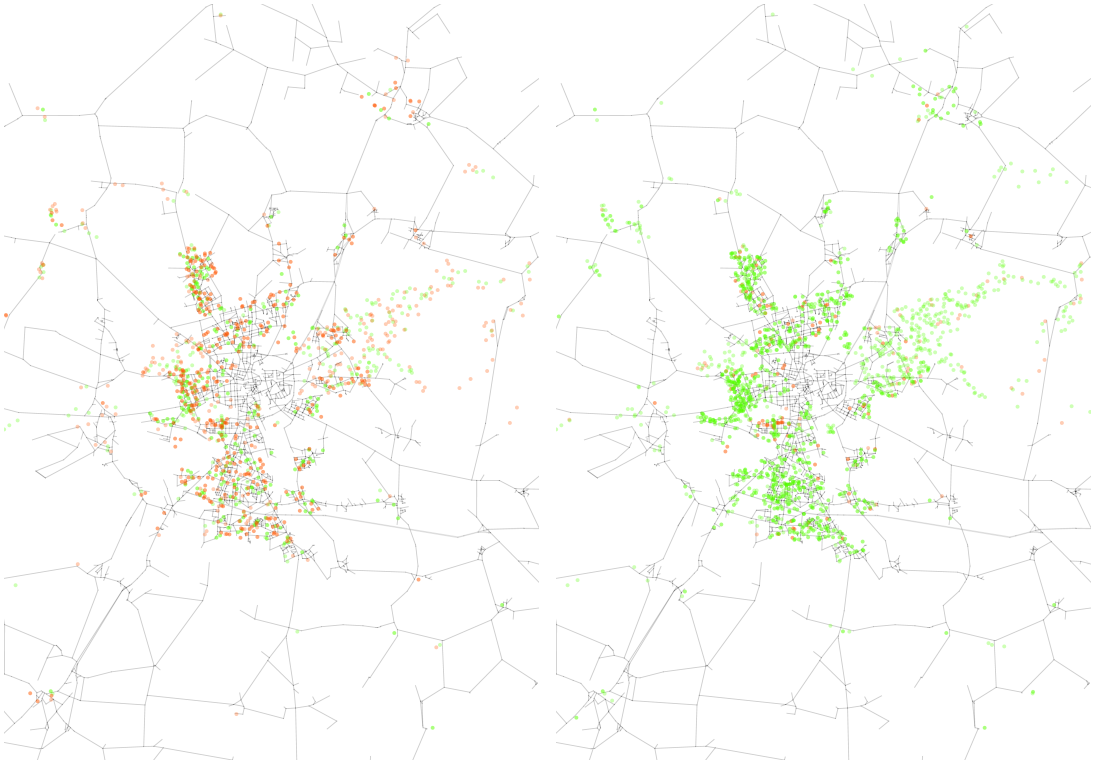


Figure 5.18.: Subnetwork constructed by the complement of the network sent to the optimization model and cut at some outer rim (black lines). The bounding box around the signalized area is depicted in light green

control is changed. Green dots belong to travelers' activities that are attracted to travel through the signalized area if signal control is changed. No clear pattern is visible. This is contrasted by Fig. 5.19b that in the same way compares the outcome of the traffic-actuated control with the base case. One can clearly observe, that the traffic-actuated control attracts travelers to travel through the signalized area.

Assignment

For the same simulations, Fig. 5.20 shows the difference between the traffic assignments of the morning peak with the base case. The more flow is assigned after a change of signal control, the more red each single link is colored. For some paths in the network the optimized coordination (Fig. 5.20a) encourages a certain direction by establishing green waves while the other direction seems to repel traffic due to a lack of coordination. The assignment for the traffic-actuated control (Fig. 5.20b) reveals stronger differences. Some links in the network are preferred to others, but there is no clear pattern.



(a) Optimization, commodities ≥ 10 (b) Traffic-actuated control

Figure 5.19.: Activity locations of travelers attracted (green) or repelled (red) by a change of traffic signal control

Spatial Dimensions

The network wide effects of signal control can be examined by the measurements of travel time, speed, vehicle kilometers, and delay as defined in Sec. 5.4.1. The time horizon for the metering is set to the full day. Besides absolute measurements, the percentaged variation of these values is calculated by

$$\frac{m_{sc} - m_{bc}}{m_{bc}},$$

whereby m_{bc} refers to a measurement of the base case and m_{sc} to the corresponding value for a change in signal control.

Full Network

First, to gather a picture of network wide reactions, the full network is inspected. If no choice dimension is enabled (Tab. 5.21a), travel time and speed decrease while delay increases for all changes of signal control. If the system is changed to the traffic-actuated control this results in a shock. For the optimized coordinations, the overall effect is, however, marginal.

(a) Optimization, commodities > 10 (b) Traffic-actuated control

Figure 5.20.: Difference of traffic assignments for the morning peak. The base case is compared to changes of traffic signal control

A different situation is observed, if route choice is enabled (Tab. 5.21b). Both optimizations result in higher travel times and vehicle kilometers, speed and delay are less than in the base case, the overall change is still marginal. In contrast to the no choice simulations, the traffic-actuated control has a positive impact on all values, i.e., less travel time, vehicle kilometers, and delay at higher speed.

Subnetworks, No Choice

These results are reflected by the flow/speed-density plots for the optimization graph subnetwork. For all plots, measurements are taken with a time interval $\tau = 5\,min$. Fig. 5.22 shows the plots for the setup where travelers have no choice. For all changes of signal control, speeds are lower at higher levels of density. The plots for the traffic-actuated control reflect the collapse of the transport system.

Subnetworks, Route Choice

If route choice is switched on, both optimizations result in less delay within the optimization graph while the delay in the city area increases (Fig. 5.21c and Fig. 5.21d). Waiting times are shifted to the outer rim of the optimization graph network. The signalized area gets less attractive for through traffic. Vehicles kilometers and travel time within the area are reduced. This is consistent with the illustration in Fig. 5.19a. Travelers avoid the optimization graph network.

The traffic-actuated control reduces delay within the optimization graph network while the delay in the complementary city area network increases. Vehicle kilometers within the optimization graph network increase and are reduced in the city area network.

Speeds are higher in both areas, travel time is reduced. This is reflected by the according flow/speed-density plots ($\tau = 5\,min$) in Fig. 5.23. Due to the improved signalization, more travelers make their journey through the optimization graph network.

simulation run	tt [hh:mm]	Δ tt [%]	v [km/h]	Δ v [%]	veh km [km]	Δ veh km [%]	delay [hh:mm]	Δ delay [%]
base case	15543:54	100.00	57.33	0.00	891166	100.00	1604:24	100.00
optimization, commodities $\geq$ 10	15626:55	100.53	57.03	-0.30	891166	100.00	1687:25	105.17
optimization, commodities $\geq$ 50	15585:06	100.27	57.18	-0.15	891166	100.00	1645:35	102.57
traffic-actuated control	29906:41	192.40	29.80	-27.53	891166	100.00	15967:11	995.21

(a) Full network, no choice

simulation run	tt [hh:mm]	Δ tt [%]	v [km/h]	Δ v [%]	veh km [km]	Δ veh km [%]	delay [hh:mm]	Δ delay [%]
base case	15480:49	100.00	57.52	0.00	890521	100.00	1571:36	100.00
optimization, commodities $\geq$ 10	15521:59	100.27	57.44	-0.08	891572	100.12	1571:19	99.98
optimization, commodities $\geq$ 50	15516:18	100.23	57.44	-0.09	891204	100.08	1563:48	99.50
traffic-actuated control	15159:27	97.92	58.40	0.87	885275	99.41	1416:25	90.13

(b) Full network, route choice

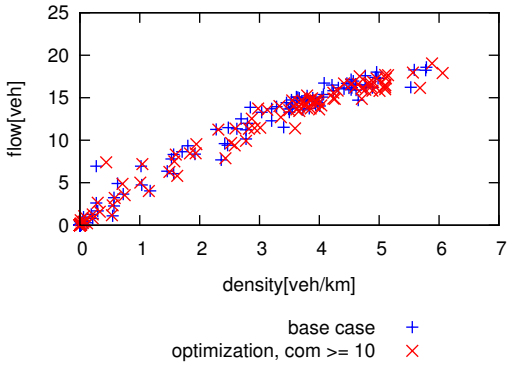
simulation run	tt [hh:mm]	Δ tt [%]	v [km/h]	Δ v [%]	veh km [km]	Δ veh km [%]	delay [hh:mm]	Δ delay [%]
base case	1972:49	100.00	30.78	0.00	60719	100.00	933:26	100.00
optimization, commodities $\geq$ 10	1919:40	97.31	31.01	0.23	59528	98.04	894:51	95.87
optimization, commodities $\geq$ 50	1938:34	98.26	30.69	-0.09	59491	97.98	918:35	98.41
traffic-actuated control	1936:48	98.17	33.77	2.99	65397	107.70	821:01	87.96

(c) Optimization graph, route choice

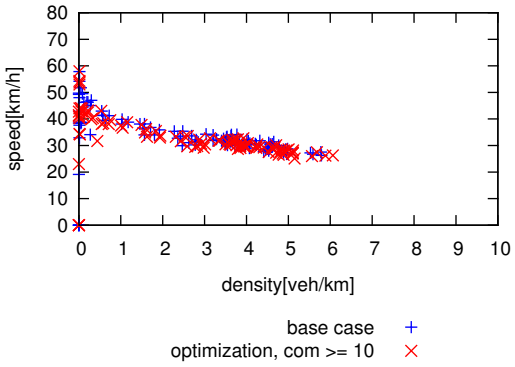
simulation run	tt [hh:mm]	Δ tt [%]	v [km/h]	Δ v [%]	veh km [km]	Δ veh km [%]	delay [hh:mm]	Δ delay [%]
base case	2444:16	100.00	46.30	0.00	113178	100.00	158:52	100.00
optimization, commodities $\geq$ 10	2500:50	102.31	45.61	-0.69	114061	100.78	181:19	114.13
optimization, commodities $\geq$ 50	2497:28	102.18	45.76	-0.54	114282	100.98	173:48	109.41
traffic-actuated control	2370:55	97.00	47.61	1.31	112884	99.74	164:41	103.66

(d) City area, route choice

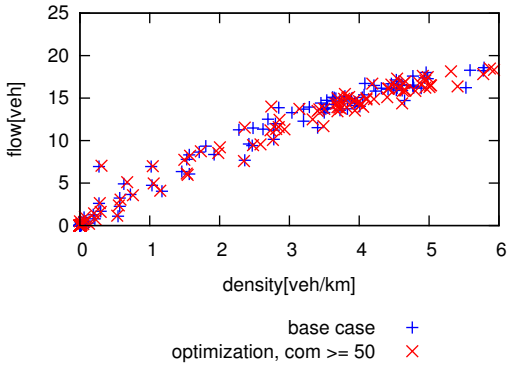
Figure 5.21.: Results for different traffic signal control strategies and (sub-)networks. Each table compares the overall travel time tt, the space mean speed v, the vehicle kilometers traveled within the (sub-)network, and the overall delay to the continued base case



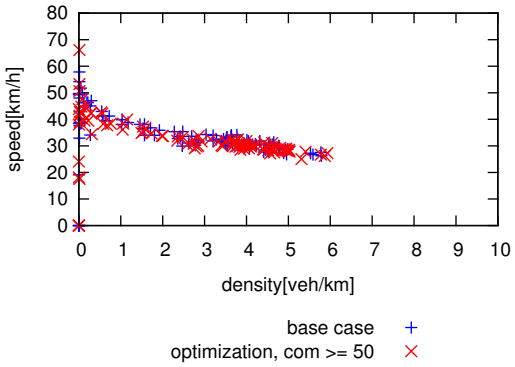
(a) Optimization, com. ≥ 10



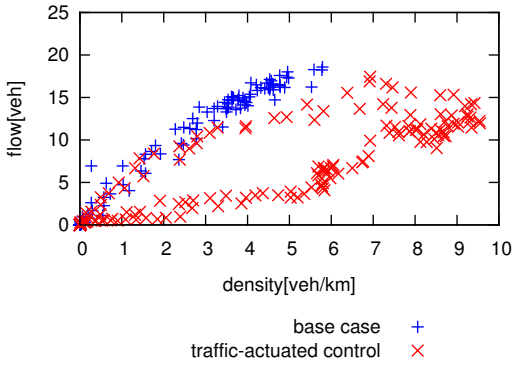
(b) Optimization, com. ≥ 10



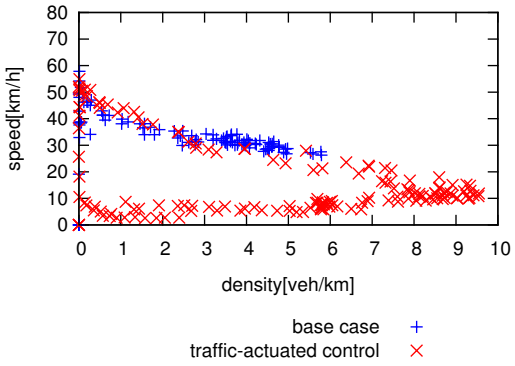
(c) Optimization, com. ≥ 50



(d) Optimization, com. ≥ 50

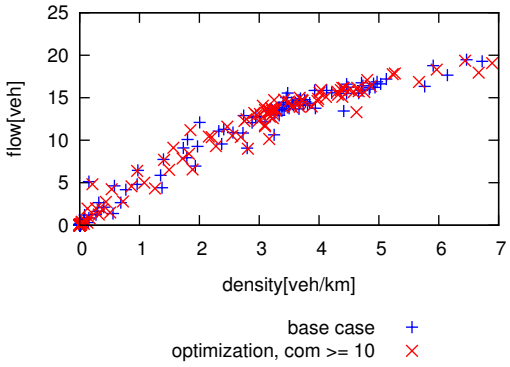


(e) Traffic-actuated control

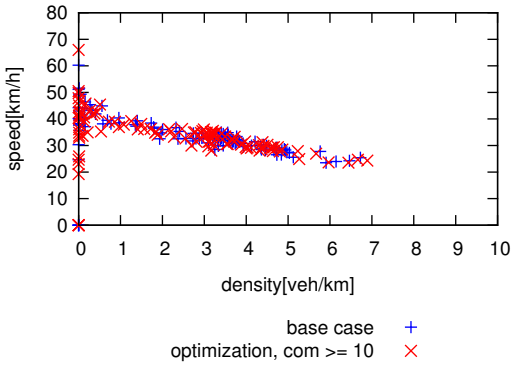


(f) Traffic-actuated control

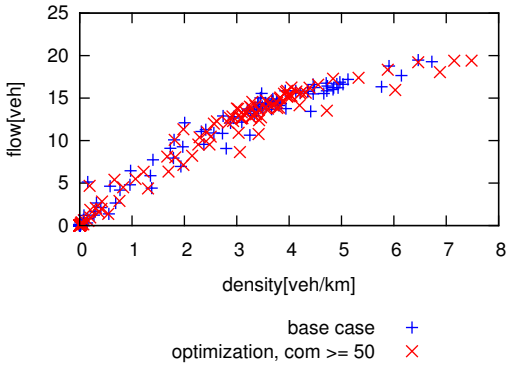
Figure 5.22.: Flow/Speed-density plots for the optimization graph subnetwork if *travelers do not react*. Each plot compares the continued base case with a change of traffic signal control



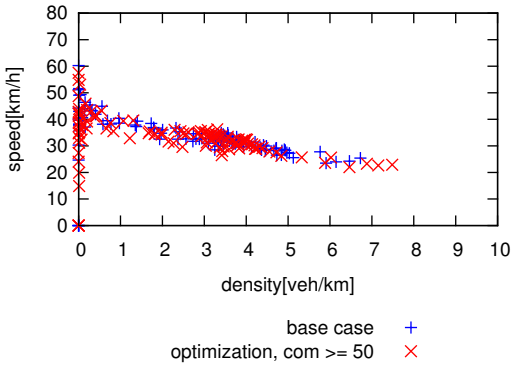
(a) Optimization, com. > 10



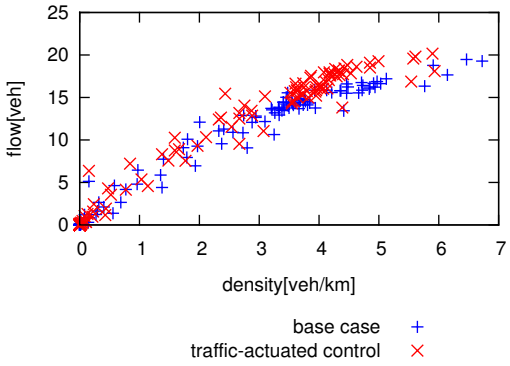
(b) Base case vs optimization, com. > 10



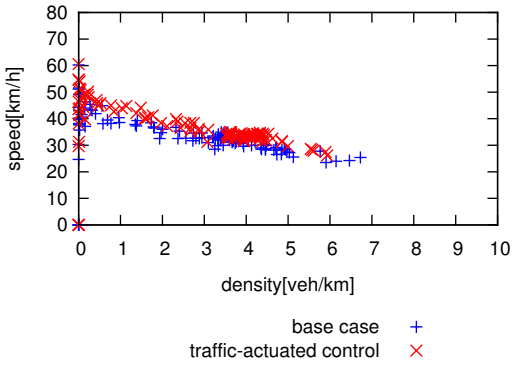
(c) Base case vs optimization, com. > 50



(d) Base case vs optimization, com. > 50



(e) Base case vs traffic-actuated



(f) Base case vs traffic-actuated

Figure 5.23.: Flow/Speed-density plots for the optimization graph subnetwork if *travelers choose new routes*. Each plot compares the continued base case with a change of traffic signal control

5.4.7. Discussion

Optimization vs Traffic-Actuated Control

First, one should point out that the comparison between the traffic-actuated control and the optimized coordination of fixed-time control is not fair. The traffic-actuated control can modify green splits. The optimization is not able to do so, yet, and changes coordination only. The comparison illustrates effects when certain parameters of signalization are changed. Results are not advanced enough to judge the different signal control strategies.

Base Case & Optimization

The base case is set up on a fixed-time control that was already optimized, but with a different demand. This provides a strong contrast to many other studies, that compare results to random green splits or coordination. The overall effects of the joint re-optimization of coordination and demand are relatively small. As expected, without route choice the system comes out slightly worse. Travelers cannot benefit from the changed coordination as they cannot change routes, the optimization, however, assumes that routes are adjusted to the improved coordination. If route choice is available, vehicle kilometers traveled within the optimization graph decrease while they increase in the surrounding area and the full network. Thus, the optimization pushes away traffic from the signalized area. The detailed representation of the demand that serves as input for optimization is not improving the overall system. The results should be complemented by a study that compares the optimized coordination to a more random coordination that is subject to current work.

Traffic-Actuated Control

The effects observed by a change to traffic-actuated signal control highly depends on the enabled choice dimensions. Without route choice, the traffic system collapses. This is an artefact of the algorithmic approximation of the adjustments that are, in practice, carefully set by traffic engineers. Extension of green time is not balanced between different approaches of a junction. If too much green time is allocated for one approach, there is only little green time left for the others. In principle, this artefact could be avoided by a manual adjustment of minimum and maximum green times. More advanced approaches for traffic signal control (e.g. Lämmer, 2007) do not need this manual calibration and might be considered for further studies. Then, an explicit definition of all red times is required. The current approach ensures by construction that all-red times are not used for green time extension. With route choice, travelers' route reaction compensates the drawbacks of the implemented traffic-actuated control algorithm. The overall system performance is improved in terms of travel time, speed, and delay. As vehicle kilometers increase for the optimization graph subnetwork and decrease for the other subnetworks under consideration, more travelers are attracted to routes through the optimization graph. There, they travel at higher speeds, with less delay.

Conversion, Boundaries

The traffic assignment produced by the optimization model (Fig. 5.16) assigns no flow to certain edges, in particular to the inbound edges at the boundaries. The origin of a commodity is a vertex of the graph that is used for optimization. The outcome of the demand conversion uses edges of the graph to define the O-D pairs. To gather commodities the downstream vertex of the O-D pair is selected as origin. This can be

changed to the upstream vertex without effort. A detailed analysis revealed that this has no effects on optimization. When the downstream vertex is unfold to the junction layout of the optimization model the commodity is inserted upstream of the junction. The traffic signals at the boundaries are thus target to optimization.

Conversion, γ

To improve conversion, the γ parameter should be adjusted to reflect the refused demand. This is not straightforward, as the overall flow of all commodities has to be smaller than the "minimal cut"[19] of the graph used for optimization. Otherwise, the problem can not be solved by the optimization. This study checks overall feasibility, the parameter γ is not adjusted.

Conversion, Optimization

The presented conversion process applies some heuristics to shrink the network to a size feasible for optimization. M. Strehler and E. Köhler (BTU Cottbus) report some progress in performance improvements of the solver that is not available, yet. As soon these improvements are available, the heuristics are no longer required and can be removed from the conversion process. Further, the same base case simulation run serves as input for optimization of morning and evening peak. Probably, similar to Rieser et al. (2008); Grether et al. (2008), the optimization of the morning peak has effects on the traffic patterns in the evening. A intermediate simulation study, that solely captures effects of an optimization for the morning peak, might be considered. Based on these results, the evening peak could be optimized with the adjusted demand.

Cottbus Scenario

The scenario for Cottbus and the surrounding area has one major drawback. There is no data for calibration available. The illustrative application in Sec. 5.3 reveals the marginal effect of signalization on travel patterns. A further calibration would improve plausibility of results. The flow-density calculations and plots help to calibrate the base case. One could calibrate further by traffic signals that are switched on and off. Then, the results from Gartner and Wagner (2004) could be taken as reference, as they rely on a more accurate traffic flow model. Overall, the Cottbus scenario is considered as nice instance. It is not too computational expensive and features many aspects of real-world transport systems. Fixed-time schedules for traffic signal control are available. Furthermore, it is based on data publicly available and could be published under a open source data license[20] as reference scenario.

Simulation Setup

The setup of simulation runs might appear complicated. Intrinsic to the simulation process is a certain amount of randomness. This may complicate analysis, e.g., when a result of iteration 1000 is compared with a result of iteration 2000. The chosen setup reduces this kind of noise in conjunction with a reasonable interpretation.

Spatial Dimensions

The subnetworks used for analysis lack in part a real geospatial interpretation. Not all links within the signalized area are covered by the optimization graph subnetwork. The network results from the conversion process to the optimization model. To improve geospatial interpretation, clearly all links should be included for analysis. The

[19] Minimal cut refers to the multi-commodity, max-flow-min-cut problem (e.g. Ford and Fulkerson, 1962) specific to the formulation of the optimization model.

[20] e.g., `http://opendatacommons.org/licenses/odbl/`, last access 17.11.2013

comparison with the complement network, however, is not affected by the geospatial inaccuracies of the networks. Complement networks are considered as good approach to analyse effects between subnetworks.

5.4.8. Summary

Overall, the simulation setup, the conversion to the optimization, and the assessment of optimization results by the simulation produces meaningful results. Network wide effects due to changes in traffic signal control can be analyzed in detail by the presented setup. It turns out, that evaluation of traffic-responsive signal control comes out completely different when evaluated without or with route choice. Yet, as stated, the optimization results have to be interpreted with care since so far they only optimize coordination but no green splits. Further, the re-optimization of already optimized coordination has only little effects, even if the demand changes. The investigations so far give no indication that the better representation of demand for the optimization lead to obviously better fixed-time control. Under re-routing, as a tendency, the optimization seems to be pushing traffic away from the controlled part of the network, while the adaptive approach seems to be pulling traffic into it.

5.5. Findings

This chapter explains, how network effects that result from changes of traffic signal control can evolve. A simulation approach for analysis of these effects is presented. The presented illustrative example explains, how traffic signal control can influence overall traffic patterns within a transport networks. A switch from a fixed-time to a traffic-responsive control may result in traffic patterns that are worse as before. Then, the modelling and setup of a real-world, large-scale scenario is presented. The scenario can be made open source and may serve as reference for other studies. It serves as basis for the preceding, preliminary results from an ongoing research project started in Feb. 2013. So far, the project has not made enough progression to gather a general conclusion. The presented results, however, illustrate, how network wide effects from changes of traffic signal control can be simulated and analyzed on a large-scale.

Chapter 6

Modeling and Simulation of Air Transport Systems

To analyze, forecast, and assess changes in air transport infrastructure and service, this chapter employs a simulation and forecasting approach for individual passenger reactions. Methods and techniques of multi-agent simulation for urban transport forecasting are used. The focus is directed on the passengers and their choice between competitive mid-distance transport modes.

The approach to represent air transport technology, i.e. aircraft, airports, and air traffic control, is based on the notion that queueing theory is frequently used to model air transport systems (e.g. Clarke et al., 2007; Hansen et al., 2009; Nikoleris and Hansen, 2012; Pyrgiotis et al., 2011) and makes use of the queue model reviewed in Chapter 3. Further, the public transit model of MATSim (Sec. 2.5) represents public transit vehicles and stops. Airports and aircraft are microscopically modeled the same way as bus stops and buses.

Passengers are represented microscopically as multi-agent demand for air transportation. Their choice of transport mode, routes, and departure time is restricted by the capacity provided by the simulation model for air transport technology. Choice modeling is kept on a low level of detail to verify overall feasibility. The chapter discusses several options to refine the approach.

After some further motiviation and backgrounds, the air technology model is presented in Sec. 6.3. Then, in Sec. 6.4, the focus is directed on the passenger demand. First, the simulation model is used as a black box. Then, the box is opened to resolve some limitations. Potential improvements and applications are discussed in Sec. 6.5. The chapter ends with a summary.

Please note that Sec. 6.3 contains and extends material published in Grether et al. (2013). Sec. 6.4 reuses and extends the preliminary results from Grether and Nagel (2013a).

6.1. Mid-Distance Transport

Rail & Air Transport

In Italy, recently a private company started providing 2.5 *h* non-stop train rides between Milano and Rome[1]. From Paris, nearly all major French cities can be reached by high-speed train in 2–4 *h* trips[2]. For the journey Berlin–Frankfurt in Germany, a 4 *h* non-stop rail connection is provided[3]. Many airlines provide flights between all these destinations that take between 1 and 2 *h*. When comparing travel times, the additional access time to the airport or railway railway station needs to be included. Overall travel times are often not that different between middle range rail on one side, and air transportation on the other.

Airport Capacity

Following recent forecasts, in 2030, 13 major EU airports will operate at least eight hours at full capacity a day (Commission, 2011). Legal opening hour constraints limit operations to a certain time frame. Yet even increasing opening hours for airports may not resolve capacity bottlenecks, since it may not be possible to move enough demand away from the peak hours.

Location & Time

In contrast, railway stations are normally not as much exposed to restrictions of opening hours due to noise protection as airports are. Also, in comparison with airports, railway stations mostly feature a more central geospatial location in urban areas. Slightly longer travel times can be compensated for by shorter access times and longer opening hours. Passenger demand and technology supply for mid-distance railway or air transportation may interact and are time dependent over a day or even a longer period.

Infrastructure Planning

To provide more capacity, railway or air transport networks may be target of planned extensions. New infrastructure is often accompanied by new emissions of noise and pollutants and is, thus, subject to lengthy planning, negotiation, and high private and public costs (Bubalo and Daduna, 2012). However, improvements on infrastructure may improve quality of journeys or offer even new possibilities of transportation. Identification and appraisal of these disadvantages and benefits is one of the key subjects in infrastructure planning.

Microscopic Assessment Model

Mutual reactions on several scales may arise if one or several transport measures cause positive or negative benefits for certain travelers. For each transport system user, changes in price, travel times, schedule, or available transport modes may have different impacts, which depend on planned activities, available budget and geospatial location.

[1] `http://www.italotreno.it` last access 19.12.2012
[2] `www.tgv-europe.com/en/`, last access 11.09.2012
[3] `www.bahn.com`, last access 11.09.2012

6.2. Models for Air Transport Systems

Simulation Tools

Many commercial simulation tools for air traffic are available, e.g., SIMMOD[4], CAST[5], AirTOp[6], RAMSrams plus[7], or Total Airspace and Airport Modeler (TAAM)[8]. All of them provide high level of detail models for airports and airspace; some of them use multi-agent architectures for different actors of the scene, e.g., for airport controllers, air traffic management, etc. Also in research, simulation toolkits of a high level of detail are available (e.g. Bilimoria et al., 2000; Sweet et al., 2002; Alam et al., 2008). All of them aim at detailed simulations of air traffic to improve air traffic management concepts. Neither commercial nor scientific simulation frameworks support agent-based modeling of individual passengers on all stages of a flight. In contrast, Clarke et al. (2007) propose an event-based simulation approach, MEANSMIT, that targets at the simulation of air traffic flow management concepts, airline scheduling and strategic planning. Passengers are explicitly included, but only during their journey. The representation of passengers enables a detailed modelling of (de-)boarding, connection flight availability, and decision making when flights shall be cancelled. MEANSMIT uses a queue model to simulate aircraft movement.

Queue Models

Queueing theory and queue models are widely used to model the technology of air transportation systems. For example, Pyrgiotis et al. (2011) use queueing theory to model the propagation of delay through the network. Effects of new airspace management technologies are studied by Nikoleris and Hansen (2012).

Chosen Approach

As queue models seem to be well suited to model air transport systems, this work applies the queue model explained in Sec. 3.2.2 in the context of traffic signal control. The model provides several parameters for an explicitly modeled segment of a transport network: The maximum flow that can pass a segment, the maximum amount of vehicles on the segment, and a maximum velocity per segment or vehicle. Several segments can be connected, building a transport network, on which individual vehicles can be simulated. Segments are modeled as FIFO (first-in first-out) queues, nodes can be interpreted as servers. Thus, the modeling of the road network is quite similar to queueing theory approaches in air transport (e.g. Pyrgiotis et al., 2011). However, the proposed model is not solved analytically but by simulation. While analytical solvable models may conserve computational resources, a computational fast simulation model enables an agent-based modeling of every individual throughout the complete simulation lifecycle in complex scenarios. For the technology side of air transport systems, the approach is quite similar to the approach chosen in MEANS—MIT (Clarke et al., 2007). It is, however, more detailed in respect to the passenger model.

[4] `www.airporttools.com`, last access 22.10.2012
[5] `www.airport-consultants.com`, last access 22.10.2012
[6] `www.airtopsoft.com`, last access 22.10.2012
[7] `www.ramsplus.com`, last access 22.10.2012
[8] `www.jeppesen.com/taam`, last access 22.10.2012

6.3. Air Transport Technology

Overview

This section focusses on the technology side of air transport networks. First, available data sources are reviewed. Then, it is shown how airports and aircraft can be modeled microscopically by a queue model based network representation and a simulation approach for urban transport systems. At the end of the section simulation studies are presented that show how the model can represent runway capacity and delay.

6.3.1. Data Sources

OAG

The air traffic technology model takes advantage of data provided by OAG Aviation[9]. An OAG snapshot of worldwide direct flights in September 2009 is available for schedule generation. All flights with IATA[10] airport codes, flight times, flight numbers and designators, aircraft types, available seats, and distance between airport are gathered from the database and processed. Codeshares, multi-stop flights, buses and trains with flight numbers, and cargo flights are filtered out of the schedule during the generation process.

Relevant data for schedule and network generation is excerpted from the OAG data using all flights departing on a Tuesday, taking each specific flight number into account only once. This may not always result in complete flight cycles, e.g., when the outbound and inbound flight operate on different days of the week. Compared to using all flights of an entire week, the network may be incomplete, as certain destinations are only served on specific days.

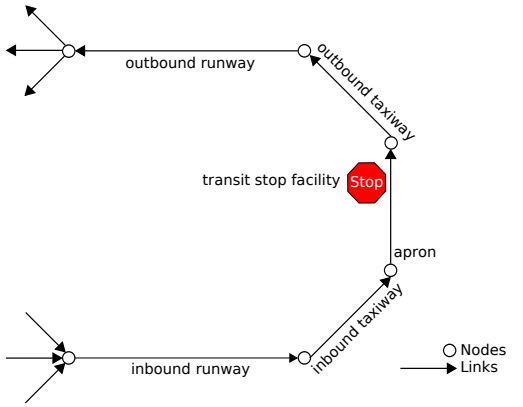
Coordinates

To enable a meaningful visualization, coordinates for airports are required. Since the OAG data does not include any airport coordinates, two alternative sources are consulted. OpenNav[11] is an online database of aeronautical navigation information featuring airport coordinates that may be retrieved with a web query based on the IATA airport code. Coordinates for those airports not available on openNav are prompted in the same manner from the Great Circle Mapper[12], which also includes a searchable database of airports. Worldwide, a total of 2683 airports with IATA code is retrieved from these data sources. The scenario used in this work contains all Europe to worldwide, non-stop flights. For this scenario 73 airports are missing in our database[13] while for the majority of 808 airports coordinates are available. Airports for which no coordinates were available were removed for the present study.

Capacity

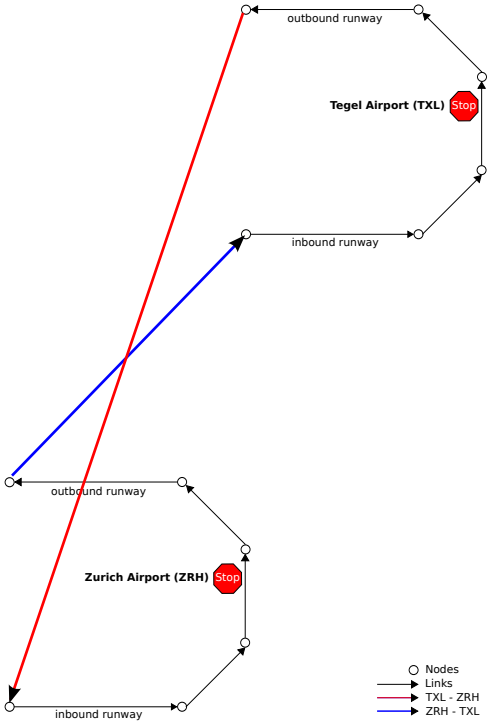
Airport capacity data is available from many sources. However, no machine-readable source was found. Thus, the 50 busiest European airports in terms of total passengers

[9] `www.oagaviation.com`, last access 08.08.2012
[10] see `www.iata.org`, last access 17.11.2013
[11] `www.opennav.com`, last access 09.08.2012
[12] `www.gcmap.com`, last access 09.08.2012
[13] Bus and train stations with IATA code are counted as missing airports when no coordinates are found.



Linktype	Count	Length [m]	Speed [km/h]
Runway	2	1500	$length \cdot c_{flow}$
Taxiway	2	500	20
Apron	1	500	20

(a) Airport layout and characteristics



(b) Network layout

Figure 6.1.: Layout of the air network

per year were taken from wikipedia[14], and data for those airports was researched manually. A list of airport capacity information is given in Appendix B, together with the source for each information item. The list provides separate capacities for departures and arrivals. All remaining airports are modeled with arrival and departure capacities of 60 aircraft per hour each. This is considerably more capacity than what these airports provide in reality.

6.3.2. Modeling

Based on the presented data an air network, a flight schedule and aircraft are generated as a precondition to run an air traffic simulation.

Network and Airports

The modelling of the air network aims at a simulation by the queue model reviewed and explained in Sec. 3.2.2. The network consists of airports, each showing an identical Network

[14] `en.wikipedia.org/wiki/List_of_the_busiest_airports_in_Europe`, last access 05.08.2012

layout, and point-to-point connections in between. Every runway is solely used either for inbound or outbound flights with taxiways connecting the runways to the apron. The latter accommodates a transit stop, i.e., the terminal, where flight movements originate and terminate (see Fig. 6.1a).

Runways

The two runways of each airport possess a restriction of flow capacity (c_{flow}) that is varied in the subsequent simulation runs. Furthermore, not more than one aircraft (unit [veh]) can be simultaneously on a runway. This is modeled by setting the $c_{storage}$ parameter of the queue model accordingly. If the flow capacity restriction (c_{flow}) should have any influence on the model the storage capacity restriction should be at least as equal to $l/v_{fs} \cdot c_{flow}$, whereby l denotes the length of the runway and v_{fs} the speed limit. If the storage capacity restriction is smaller, flow constraints would not have any effect. As both values flow and storage capacity shall be set, the speed limit is varied according to the chosen value for flow capacity. E.g., for an outbound runway of an airport with an outbound flow capacity of $60\,veh/h$ on a $1500\,m$ runway with a storage capacity constraint of $1\,veh$, the speed limit is set to $\frac{1500\,m \cdot 60\,veh/h}{1\,veh} = 90\,km/h$.

Airways

Each airport pair is directly connected by airway links, one for each flight and direction of travel (see Fig. 6.1b). The maximum speed on any of these links is calculated based on the distance and flight duration provided by OAG. Times for taxi, take-off, and landing are also taken into account, i.e., the flight duration is reduced by the time needed from push-back to airborne before the maximum speed for an airway link is calculated.

Routing

To simplify matters, ATS (Air Traffic Services) routes are not implemented. Further, despite data could be gathered, no permission for use is retrieved so far. Note however, that each flight has an individual link that could be interpreted as route, each possessing individual characteristics. Fig. 6.2 shows parts of the network for simulation of European air traffic.

Flight Schedule

Schedule

The flight schedule is taken from the OAG data and translated to a MATSim *Transit Schedule* (Sec. 2.5) containing information about each line, route, and departure. For each airline that offers a connection between two airports, a transit line is generated. A transit route, which represents the route on the air traffic network, is created for each flight offered by this airline. The route contains the links belonging to the airport representation plus the specific link for this flight connecting the airports' out- and inbound runway. After take-off, mutual interferences of aircrafts en-route are not included in the model. Tab. 6.1 lists the number of (not included) airports, direct origin-destination (O-D) connections and flight movements for three different area pairs.

Time Zones

For matters of consistency, all local times are converted into Coordinated Universal Time (UTC). This ensures aircraft taking off and landing at the scheduled times throughout all time zones and also enables the model to reflect incoming and outgoing waves at hub airports worldwide at the appropriate times.

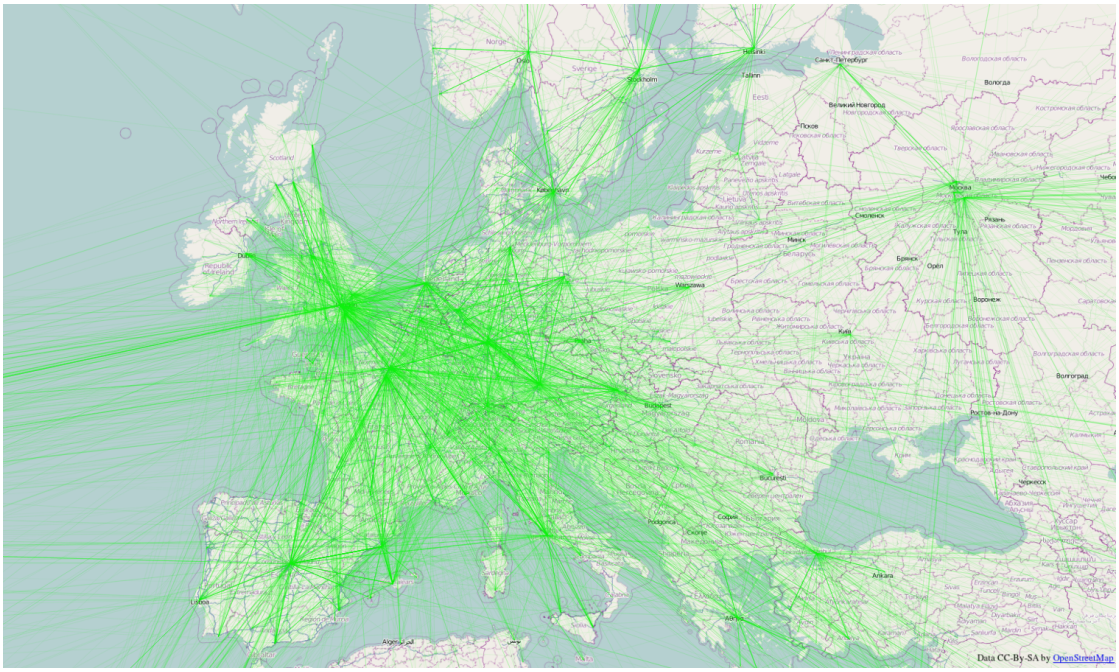


Figure 6.2.: European air network with country borders in the background (country borders © openstreetmap.org)

Aircraft

To represent individual aircraft in the simulation, transit vehicles are created on the basis of OAG data IATA aircraft codes, operating airlines, and seating capacities are reflected in the respective aircraft representation for every flight. Information about boarding times, i.e., passenger flow per door over time, is not available, but could be set for each aircraft type. One aircraft per flight is generated, thus delays resulting from a delayed incoming aircraft are not modeled. Accordingly, no aircraft rotations and vehicle trip chains are implemented for the time being. The maximum velocity of each aircraft is set to twofold sonic speed, since speed limitations are set for each airway link of the network. Vehicles

6.3.3. Results

Results of a simulation for flights to, from, and within Europe (referred as "Europe to worldwide" in Tab. 6.1) are presented in the following. Several versions of the model are simulated allowing a comparison of a model without capacity constraints, a model with runway capacity constraints, and a model including some delay. The simulation Overview

Area pair	Airports	Airports missing	O-D Pairs	Flights	Flights missing
Worldwide	2333	81	27496	56376	2644
Europe to worldwide	808	16	14156	21425	577
Germany to worldwide	269	3	2814	4394	189

Table 6.1.: Numbers for different geospatial extents of the model

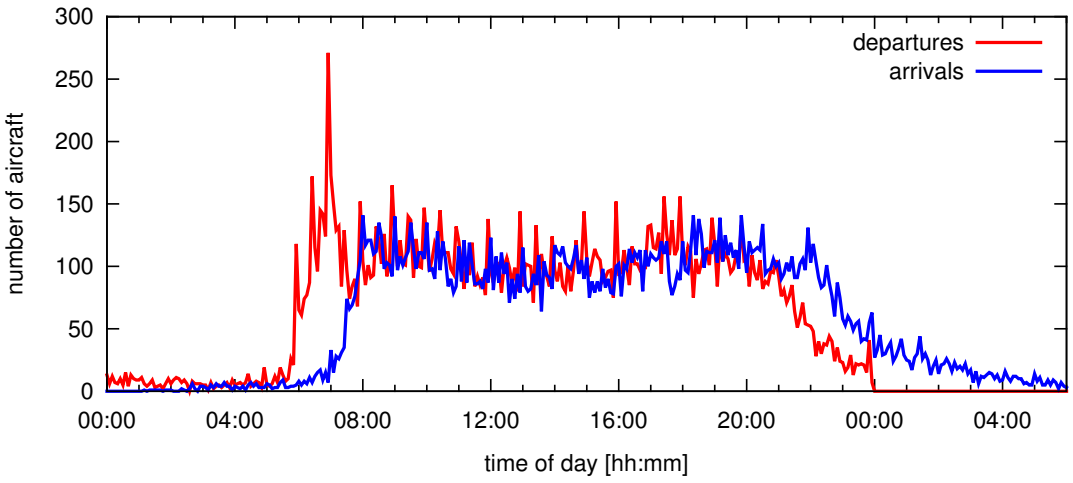


Figure 6.3.: Arrivals and departures over time of day. Europe to worldwide model with high runway capacities.

is run for one iteration which starts at midnight and continues until after 40 h the last flight has arrived at its destination.

3 Experiments

Delay resulting from changes of runway capacity is studied in three experiments. Scheduled flight times from the OAG data are compared to the simulated time each flight needs from gate to gate, i.e., its departure transit stop facility to its arrival transit stop facility. The resulting arrival delay distributions are shown in Fig. 6.4.

Unlimited Runway Capacities

First, the simulation is run with unrealistically high runway capacities. As expected, all flights are on time. Fig. 6.3 shows the simulated number of departures and arrivals over time of day. Clearly, one can observe the morning departure peak between 05:00 and 07:00 UTC. The resulting delay distribution is depicted in Fig. 6.4a.

Uniform Capacity 60/h

Second, in order to test sensitivity, all runway capacities are set to 60 veh/h , i.e., on each runway one take-off or landing per minute is possible. This is effectively *larger* than in reality for most airports, except for Paris, Charles de Gaulle (CDG) and Amsterdam, Schiphol (AMS), where it is less (Appendix B). The impact on the system is

more profound than one might expect, the delay distribution is shown in Fig. 6.4b: 10589 flights, i.e., 49.2 % of the simulated 21425 flights, arrive at their scheduled time; 99.6 % of the flights possess less than 16 min delay. The most delayed flight arrives 28 min after scheduled arrival.

Real capacity

In the third experiment regarding delays, airport specific data is used for capacity of runways (Sec. 6.3.1, Appendix B). Each modeled airport's arrival runway is set to the arrival capacity from the table, and each departure runway is set to the departure capacity from the table. If no data is available, each runway's capacity is set to 60 veh/h as in the previous experiment; since these are fairly large capacities, this implies that the model will generate few if any delays at those airports. Fig. 6.4c shows that the resulting overall delay distribution is similar to the simulation run which was based on homogeneous runway capacities (Fig. 6.4b). 45.5 % of all flights arrive at the scheduled minute, while 99.2 % have a delay less than 16 min. The latest arrival is 32 min beyond schedule.

Other Reasons for Delay, Normal Distribution

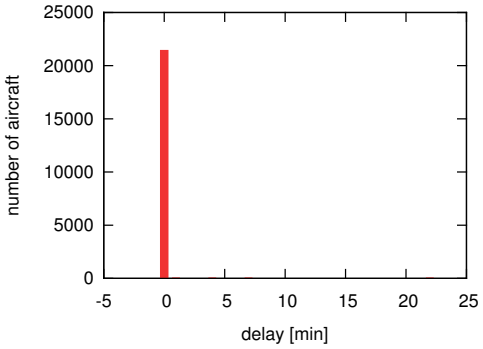
Limited runway capacities are a source of delay (Bubalo and Daduna, 2012). The presented experiments show that the model can capture these delays. By themselves, however, restricted runway capacities can only explain a small part of overall delay. For 2011, the Central Office for Delay Analysis (CODA) reports that 37.1 % of all flights were delayed on departure (Central Office for Delay Analysis, 2012), with an average delay of 27.6 min. Those CODA values for 2011 are used for a rough approximation of randomly occurring delay in the simulation, as follows. In a preprocessing, a 37.1 % sample of all simulated flights is drawn, using a uniform distribution. The length of delay is then drawn from a normal distribution with a mean of 27.6 min and a standard deviation of 13.8 min, and added to the scheduled departure time. In order to get a clear picture of the effects of this method, the simulation is first run without capacity constraints. The resulting overall delays are shown in Fig. 6.4d; delayed flights show the expected shape of a normal distribution around 27.6 min. For the next simulation run, the model with airport specific runway capacities is simulated jointly with the random delay. About 39 % of all flights are now delayed more than five minutes, the average delay is 27.90 min. The resulting overall delay distribution (Fig. 6.4e) still possesses the shape of the normal distribution, but effects of runway capacity restrictions are observable as well.

Other Reasons for Delay, Gumbel Distribution

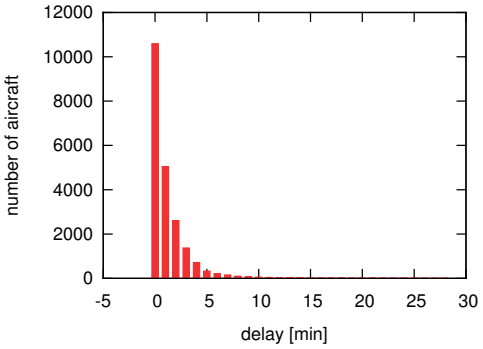
In Figs. 6.4e and 6.4d, delays are generated by first deciding if an aircraft is delayed, and then generate a delay as a normal distribution with mean 27.6 min and standard deviation 13.8 min. This model stems from the definition of delay used by the operators. It produces, however, implausible results in the sense that it makes delays of about 10 min less probable than delays of around 30 min. In consequence, another model is tried where arrival times are distributed around a delay of 0 min. Since late delays are more probable than early delays, an asymmetric distribution was chosen, namely the Gumbel distribution with a cumulative distribution function

$$F(x) = e^{-e^{-(x-\mu)/\beta}}$$

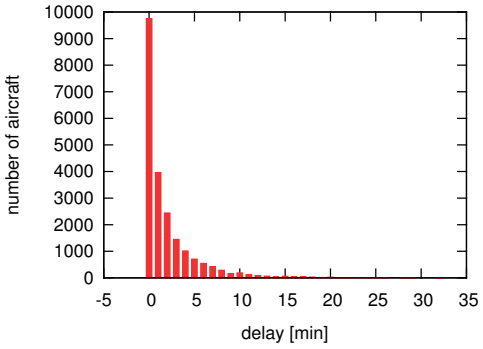
and parameters $\mu = 0$ and $\beta = 6.47\,min$. Results for this model including airport specific runway capacities are shown in Fig. 6.4f. The shape of the simulated delay distribution appears quite similar to the delay distribution shown in the CODA report (Central Office for Delay Analysis, 2012, Fig. 13). For a detailed statistical analysis the raw data of the aggregated values presented in Central Office for Delay Analysis (2012) is required.



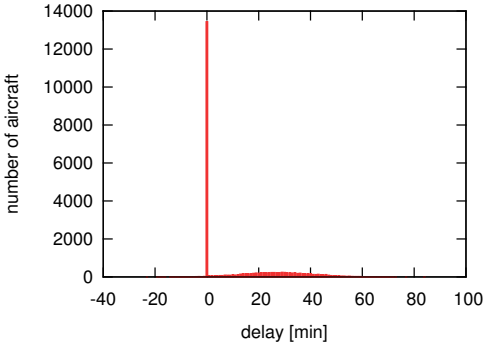
(a) High runway capacities



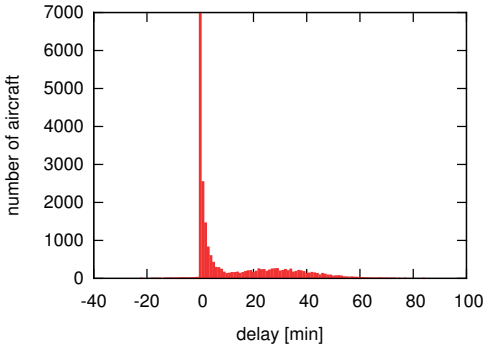
(b) Uniform runway capacities of 60 *vph*



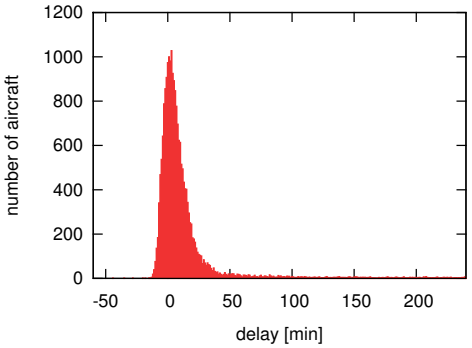
(c) Airport specific runway capacities



(d) Normal distribution & high runway capacities



(e) Normal distribution & airport specific runway capacities



(f) Gumbel distribution & airport specific runway capacities

Figure 6.4.: Results for different delay models. Number of delayed aircraft over minutes of delay.

6.3.4. Interpretation & Discussion

The results show that the proposed approach can model some important characteristics of air traffic technology. In particular, runway capacity restrictions can be added to the model.

Runway Layout

The current model uses two separate links for the runways of an airport, one for arrival and another for departure. In reality, it might happen that both runways are used for the same purpose for periods of time. The model could possibly be improved by modeling both runways as one link. When doing this, however, more elements of air traffic control would need to be included, such as prioritization between incoming and outgoing aircraft. Furthermore, values for capacity and speed of the runway have to be adjusted.

En-Route Delay

Also, the model of the air network is not capturing delays resulting from en-route capacity constraints that may occur in the real ATS route network. Due to differences in air traffic flow and capacity management strategies, the present model may be more appropriate to model US airspace than EU airspace. Following Lulli and Odoni (2007), en-route delay in the US airspace accumulates at the destination airport while the European air traffic management captures delay already during the flight. For the European airspace the time dependent network attributes feature of the simulation (Sec. 3.1.2) that can vary a link's flow capacity, or set speed limits for certain time periods could be applied to get a more realistic representation. Finally, the ATS route network itself could be included in the modeling process if exact data and routes are provided.

Reactive Delays

Reactive delays due to delayed incoming aircraft are not reproduced as aircraft rotations are currently not included in the model. The multi-agent approach is, in general, particularly suitable to model reactive delays. One would either need detailed trip-planning and scheduling data from private companies, or appropriate approximation algorithms for these elements to include reactive delays. The modeling itself is then straightforward.

Other Reasons for Delay

Not explicitly modeled reasons for delay can, in principle, be captured by draws from an appropriate probability distribution. The probability distribution hinges clearly on the data available for calibration and the explicitly modeled sources of delay. Because of the lack of detailed data for delays, statistical analysis is limited. The presented results, however, show two options how not explicitly modeled reasons for delay can be captured.

6.4. Passenger Demand

With the results from the previous section, an air transport technology model is available. This section shows how a passenger demand for air transport can be modeled on top of the technology with the multi-agent approach.

6.4.1. Data Sources

Data Providers

There are many different ways in which passenger demand for transport systems can be generated (e.g. Balmer, 2007). One option is to start with origin-destination (O-D) flows between geographical regions. In a European context, possible data-sources include OAG Aviation[15] and eurostat[16]. They provide data about passengers; O-D flows, however, are not provided. Data-sources geographically limited to Germany as in "Der Flughafenverband"[17] or ITP/BVU (2005) do not come with O-D data, neither. The latter may have O-D relationships available in an upcoming version. The German Institute of Air Transport and Airport Research (DLR, Institut für Flughafenwesen und Luftverkehr) provides monthly statistics containing O-D flows[18], but the pdf-format provided is not suited for machine reading, and data is only available up to 09-2010. DESTATIS[19] provides O-D data by airport for German air traffic in a machine readable format. Data is available for whole years or a specific month. DESTATIS data is thus used in the following to create an agent-based air transport demand for Germany.

DESTATIS

The passenger demand is based on the data for 09-2009 in order to be consistent with the flight schedule of the air transport technology model (Sec. 6.3). DESTATIS provides data for passenger movements within Germany in two different representations (referred as data sets 5.1.1 and 5.1.2). The number of O-D trips between airports is captured in two different ways. For all pairs of airports, the number of direct trips between the airports is given in the data set 5.1.1. Furthermore, the second data set, 5.1.2, contains O-D pairs that do not include transfers, but the final destination. E.g., one person flying from Hamburg (HAM) via Frankfurt (FRA) to Munich (MUC) is contained in the data set 5.1.2 as one O-D pair: HAM $\rightarrow$ MUC. The 5.1.1 data counts this person twice, once on the O-D pair HAM $\rightarrow$ FRA and once on FRA $\rightarrow$ MUC.

Passengers, Synthetic Population

The passenger demand of the second data set, 5.1.2, is used to create the synthetic population. For each O-D pair the number of trips is scaled from monthly to daily values by a division by 30. Then, for each O-D pair and trip a virtual person is created. The resulting synthetic population contains 51832 virtual persons, 1550 trips from the original data are neglected as origin and destination are equal. Each virtual person performs two activities, one at the origin and the other at the destination airport. Both activities are of same type, thus time spent performing both activities is accumulated before it is evaluated by the utility function according to equation (2.3). A *typical duration*, t_*, of 21 h is set for this activity type. In between the two activities a flight leg is scheduled, connecting origin and destination. As is common, the demand does not specify if a direct flight from O to D is chosen or the virtual person is on a route containing one or

[15] `www.oagaviation.com`, last access 08.08.2012
[16] `ec.europa.eu/eurostat`, last access 10.09.2012
[17] `www.adv.aero`, last access 10.09.2012
[18] `http://www.dlr.de/fw/en/desktopdefault.aspx/tabid-2961/9753_read-19683/`, last access 10.09.2012
[19] `destatis.de`, Fachserie 8 Reihe 6, last access 10.09.2012

more transfers. The time virtual persons arrive at the origin airport and start waiting for a connection is drawn randomly from a uniform distribution in 04:00 to 18:00, UTC. This reflects estimated typical opening hours of airports in Europe.

6.4.2. Simulation Setup

Parameters

The synthetic population serves as input for the simulation. As scenario for air transport technology, the Europe to world wide model with no delays and no effective runway capacity restrictions from Sec. 6.3 is used. The assignment of concrete flights to the desired O-D connection, i.e., the passenger routing, is calculated by the default public transit routing module of MATSim (Sec. 2.5). The routing basically looks for a least cost path in terms of travel time. The graph used for routing is constructed from the information contained in the transit schedule. Each flight is represented by an edge. Transfers are modelled by additional edges, that are implicitly added to the graph. To penalize transfers, the routing assumes an additional cost of $c_{lineswitch}$ for each transfer edge. The same parameter is also considered by the scoring function, i.e., a (dis-)utility of $-c_{lineswitch}$ is added to the score of a virtual person for each transfer. The simulation is run several times using different values of the $c_{lineswitch}$ parameter, i.e., $c_{lineswitch} \in \{0, -6, -12, -18, -24, -30\}/transfer$[20].

Simulation Runs

Each simulation is run for 600 iterations. In each iteration, 10 % of the virtual persons may shift their departure time randomly within a $2\,h$ interval. The amount of shift is drawn from a uniform distribution. Another 10 % may seek a new route, i.e., a connection between origin and destination. Each passenger chooses out of a set of 5 plans using the multinomial logit model (Sec. 2.1). The outcome is stable after 500 iterations, thus departure time choice and routing are switched off. For another 100 iterations only the logit model is used by the passengers to select a plan. Empirically, fixing the choice set for the last 100 iterations reduces the noise of learning and eases analysis and intepretation of results.

Computation Time

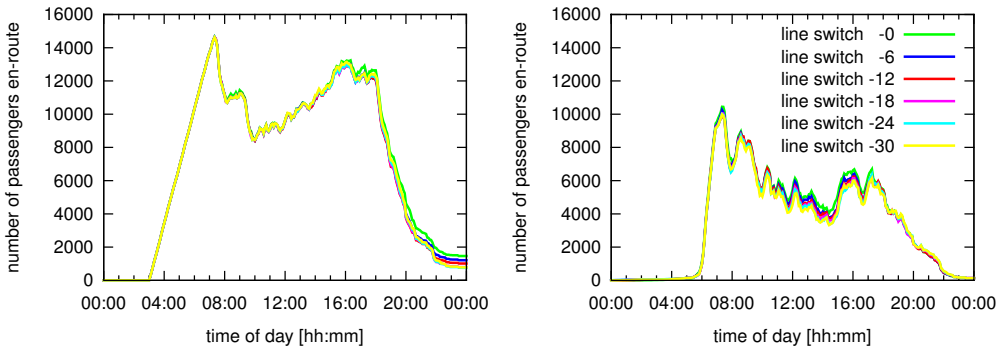
One iteration takes around $5\,min$ on an Intel Xeon Processor ($2.67\,GHz$) using one core for the execution of mobility simulation and two cores for the replanning modules. Overall computation time for one simulation run is roughly 2 days.

6.4.3. Results

Zeroth Iteration

First, to show the effects of routing, the result after the zeroth simulated iteration is presented. Each virtual person gets a connection assigned based on a generalized cost routing for the connection and the preset departure time. Fig 6.5a shows the number of travelers en-route, i.e., waiting for a flight or traveling by plane, as a function of the time-of-day. Some passengers are still waiting for a flight at midnight. As only

[20] Note, that $c_{lineswitch}$ cannot be set to values > 0 as a standard least cost path calculator cannot handle positive costs for edges.



(a) After 0th iteration (b) After 600th iteration

Figure 6.5.: Passengers waiting for a flight or traveling by plane over time of day

one day of operation is simulated, these passengers are stuck and will not reach their destination. The number of these *stuck* passengers is decreasing with the increasing disutility of line switch.

Iteration 600

The output after 600 iterations is depicted in Fig. 6.5b. The shape of all curves is different from the shape of the zeroth iteration. One can identify two morning and two evening peaks. Some passengers still get stuck at the end of the day, but fewer than in the 0th iteration. In addition, the influence of the $c_{lineswitch}$ parameter is diminishing in respect to the stuck passengers.

Error Calculations

To study the influence of the $c_{lineswitch}$ parameter, the simulation results are compared with the input data. Recall, that the synthetic population is generated based on O-D pairs that may contain transfers ($od_{transfers}$), while other data directly counts the number of passengers on actual direct flights (od_{direct}). The latter is used to evaluate the accuracy of the model. For comparison, the number of passengers on direct flights is calculated for each O-D pair (sim_{direct}) from the simulation results.

Based on these data sets, the mean square error σ^2 is computed as

$$\sigma^2 = \frac{\sum_{i \in OD} (sim_{direct}(i) - od_{direct}(i))^2}{|OD|},$$

whereby $|OD|$ denotes the number of O-D pairs, $sim_{direct}(i)$ the simulated passengers on a direct flight between the O-D pair i, and $od_{direct}(i)$ the same, but retrieved from data. With the same values, the (unsigned) mean relative error for each O-D relation is calculated as

$$\text{mean rel error} = \frac{\sum_{i \in OD} |(sim_{direct}(i) - od_{direct}(i))| / od_{direct}(i)}{|OD|}.$$

Results

Tab. 6.2 shows the results for these calculations. The first line contains the comparison

of the two sets of input data from DESTATIS, i.e., in the above formulas, sim_{direct} is replaced by $od_{transfers}$. This serves as reference as it would assume that *all* demand is served by direct flights. All simulation runs explain the data better than that reference. Mean square error, variance, and number of stuck passengers increase with decreasing values of $c_{lineswitch}$. The relative error, however, decreases. So far, variance and relative error are hard to interpret, as they point in opposite directions. First, the reasons for stuck passengers are analyzed in detail.

$c_{lineswitch}$	σ^2	σ	mean rel error	stuck
$od_{transfer} - od_{direct}$	12640	112	1.75	-
-0	9293	96	0.36	320
-6	9878	99	0.35	338
-12	10361	102	0.33	350
-18	10552	103	0.32	378
-24	10916	104	0.32	373
-30	11090	105	0.32	386

Table 6.2.: Simulation results for different values of $c_{lineswitch}$, iteration 600

Passengers Stuck

Some passengers fail to reach their destination, they get stuck. This is considered unrealistic, as only trips within Germany are modeled, which are usually completed within a few hours without any requirement for an overnight stay at an airport. Filtered by flights in Germany, Fig. 6.6a depicts passengers in aircraft and seats therein over time of day. Getting stuck is not a consequence of a general lack of seats: at any time of day, there are more seats than demand. There are many reasons why stuck passengers can arise in such a situation. Further analysis of the simulation results leads to the following insights for the $c_{lineswitch} = 0$ scenario:

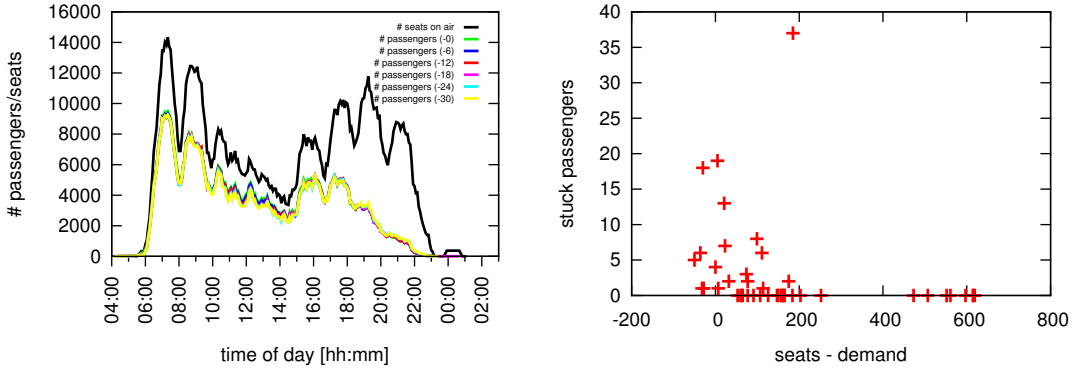
- 92 passengers are stuck because there is no seat, *and* there is no other flight by the same airline later during the day to which they would be shifted otherwise.
- 228 passengers are stuck at an airport because there is no connection after their departure time between that airport and their destination airport.

Departure Time

To study the influence of departure time on available connections, several simulations are run that set the departure time of each passenger being stuck to 04:00 UTC, i.e., before the first aircraft is departing. Simulation results produced similar findings as presented above.

Demand & Seats

Thus, it is worth looking more closely at the relation between passengers being stuck and the capacity of seats offered for each O-D pair. For each O-D pair, one can obtain the number of travelers that plan to travel from O to D, i.e., the number of virtual persons, $od_{transfer}$ (data set 5.1.1). Further, the number of seats offered on that O-D pair can be retrieved from simulation input data. Fig. 6.6b plots the number of travelers that are stuck on their planned O-D connection over the difference between seats and $od_{transfer}$.



(a) Passengers in aircraft and available seats over time in Germany, iteration 600

(b) Correlation between the available seats, the demand for seats ($od_{transfer}$) and the number of passengers being stuck, iteration 600

Figure 6.6.: Potential reasons for stuck passengers

To improve visibility the figure is cut at values where available seats increase demand by more than 800 — the number of stuck persons is always 0. Apparently, passengers get more likely stuck the more the requested demand is equal or greater than overall capacity.

6.4.4. Adding an Alternative Mode

Simulation Setup

To gain further insights, in the following a slightly different simulation setup is applied. The additional cost for each transfer is fixed to $c_{lineswitch} = 0$ and has no influence on the model. Instead, a second option for mode choice is added. Each virtual person can now choose between the micro-simulated air transport options and an alternative mode. The alternative mode has no capacity restrictions. Furthermore, passengers that travel with the alternative mode can start directly at their desired departure time. The travel time, tt, is computed by the microsimulation with an estimation of the beeline distance between the O-D pair d and a velocity v, i.e., $tt = d/v$. This velocity is varied in several simulation runs, i.e., $v \in \{100, 150, 200, 250, 300\}[km/h]$. If the alternative mode is chosen, the (dis-)utilities for traveling in the scoring are calculated accordingly.

Each person in the synthetic population obtains a second plan that uses the alternative mode. With this population the simulation is again run for 600 iterations. Like in the previous simulations 10 % of the virtual persons may shift their departure times while another 10 % seek a different route between origin and destination in the air transport network. Additionally, further 10 % of virtual persons may change mode, i.e., they can switch between the air traffic mode and the alternative mode. After 500 iterations all

$v[km/h]$	σ^2	σ	mean rel error	stuck
$od_{transfer} - od_{direct}$	12640	112	1.75	-
100	9388	97	0.37	67
150	9911	100	0.35	50
200	12075	110	0.37	6
250	13759	117	0.39	0
300	13790	117	0.42	0

Table 6.3.: Simulation results including an alternative mode at different speeds v

$v[km/h]$	# air mode	# alt. mode	# stuck	air mode[%]	alt. mode[%]	stuck[%]
100	51143	622	67	98.67	01.20	00.13
150	50213	1569	50	96.88	03.03	00.10
200	48541	3285	6	93.65	06.34	00.01
250	46748	5084	0	90.19	09.81	00.00
300	43698	8134	0	84.31	15.69	00.00

Table 6.4.: Modal split for different speeds of the alternative mode, iteration 600

choice modules are switched off, thus for the last 100 iterations the logit model is used by passengers to select one of their plans.

Results

From the output of the 600th iteration, the same numbers as for the previous simulation runs are calculated (Tab. 6.3). If the speed of the alternative mode is 100 or 150 km/h mean square and relative error are quite similar to the previous results. The number of stuck passengers, however, is remarkably reduced from approx. 320 to 67. Alternative mode speeds higher than 150 km/h further reduce the number of stuck passengers. Both error values increase. As passengers no longer get stuck, the model seems more plausible, but deviates from the given data.

The increasing speed of the alternative mode affects the modal split (Tab. 6.4). While for a $v = 100\,km/h$ the alternative mode is chosen by 1.2 % of the passengers, a mode alternative with a speed of 300 km/h attracts 15.69 % of travelers.

Fig. 6.7 illustrates temporal effects for the alternative mode at speeds of 100 km/h and 300 km/h. One can observe that passengers using air transport follow the time distribution of the offered capacity. In contrast, travelers on the alternative mode are spread over time of day. This is plausible considering the setup of simulation: Passengers have no time constraints that force them to arrive at a certain time at their destination. Departure times are equally distributed between 04:00 and 18:00, UTC, and then randomly mutated during the iterations. As the alternative mode is always available there is no constraint within the model that ties passengers to any departure time.

Black Box?

One might conclude with these results. A more accurate inspection of the results, how-

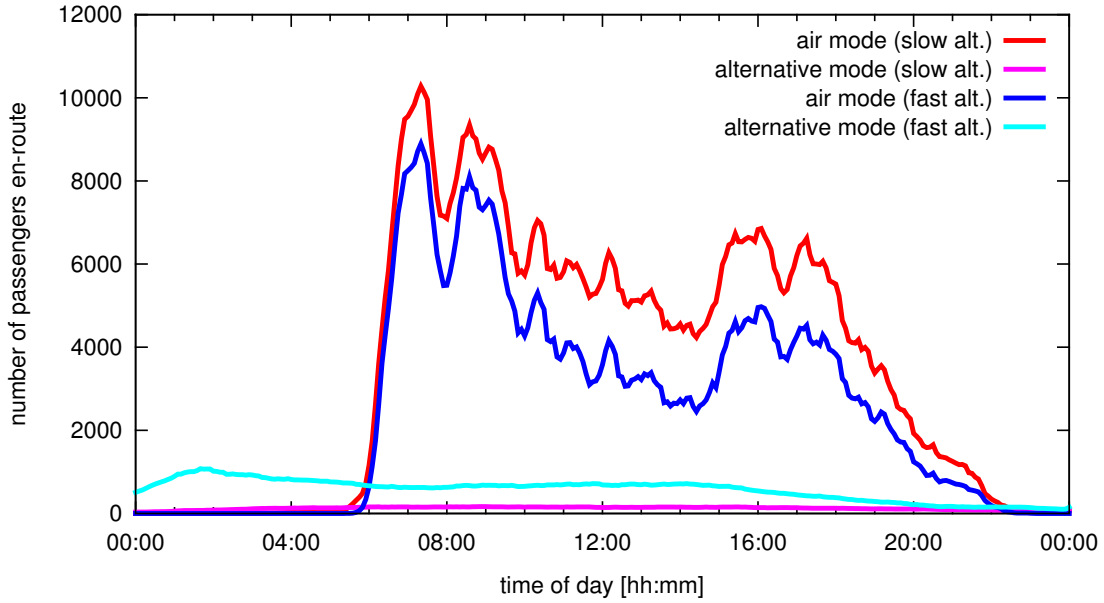


Figure 6.7.: Passengers waiting for a flight or traveling by plane or by the alternative mode over time of day, iteration 600

ever, raises the question why travelers on both modes have a tendency to depart early in the morning. Furthermore, if a not capacity constrained alternative is available, one might wonder why travelers prefer to get stuck instead of traveling on the alternative mode. One can reveal further insights by a closer look at the implemented algorithm.

Time Choice

The public transit used for the simulation runs boards passengers to transit vehicles in order of appearance at the transit stop. If the passenger capacity of the vehicle is reached, passengers can no longer board. If there is passenger capacity available in a subsequent transit vehicle of the same line, passengers may board this vehicle (Rieser, 2010, p. 67). For the air transport model, this implies that the earlier travelers arrive at the airport, the higher is the probability they get a seat. This explains the passengers' tendency towards earlier flights, even though the modeling is not specifying any constraints that justify this behavior. To understand the shift towards earlier departures in case of the alternative mode, a closer look at the mode choice model is required.

Mode Choice

The mode choice model tested and validated in (Rieser et al., 2009) specified a constraint for the plan database: Only one mode can be set for each plan. At least one plan of every mode is kept in the plan database of every virtual person. This constraint was removed in further studies, motivated by the need of more complex plans. In the current version, each leg between two activities can have a different mode. The mode of each leg is then varied randomly over the iterations (Rieser, 2010, p. 77).

Implications

Aware of these changes, one can now explain why travelers get stuck even if there is

an alternative. Slow speeds of the alternative mode implicate a dominance of the air transport mode. If there is a seat on a flight, travelers receive a higher score than by traveling on the alternative mode. For the first 500 iterations, some virtual persons try different routes, times, or modes. Thus, during this phase a certain amount of seats is not occupied. More virtual persons validate the air mode as good choice than there are available seats. The corresponding plans get a higher score than the plans for the alternative mode and shift towards earlier departure times. Recall, that the plan with the lowest score is deleted when the number of plans excesses a certain threshold (Sec. 2.1). For the first 500 iterations, this is the plan for the alternative mode. If plans for the alternative mode are recreated, they are copied from an air mode plan and obtain its departure time. Then, choice dimensions are switched off. Only the logit model is used for selection of existing plans. Travelers, that have tried the alternative mode in the previous iteration, switch back to the air transport mode with a high probability. This results in a lack of seats that are then allocated by time of arrival. Passengers rejected to board their flight get stuck. The probability to avoid getting stuck is higher the earlier one arrives at the airport. As the plan for the alternative mode is deleted before air transport plans, in most cases the plan database of a virtual person no longer contains this option. Otherwise, passengers may switch to the alternative mode with a tendency towards an early departure.

Potential Solution, Stuck

This analysis reveals several problems to be solved in the overall simulation process before air transport specific questions can be addressed further. In the following, a potential solution is presented that prevents passengers to get stuck. Theoretically, one could increase the threshold for the maximum number of plans per person to infinity. In practice, this is not feasible, as memory is limited. The solution used for the study in (Rieser et al., 2009) cannot be applied to more complex plans. Some more complex heuristic is required to measure the similarity of plans. This heuristic can be implemented by a functionality similar to the path size logit formulation (e.g. Frejinger and Bierlaire, 2007). That is, a score penalty, $\beta_{PS} \cdot \ln PS_{in}$, can be added to a plan when it is similar to other plans, whereby β_{PS} is a scale parameter and PS_{in} measures similarity by

$$PS_{in} = \sum_{a \in \Gamma_i} \frac{l_a}{L_i} \frac{1}{\sum_{j \in C_n} \delta_{aj}},$$

whereby Γ_i is the set of legs in plan i, C_n the set of plans of person n, and δ_{aj} the overlap function. δ_{aj} equals 1 if leg a is contained in plan j and 0 otherwise[21]. The scheduled travel time of leg a is l_a, and L_i denotes the sum of all scheduled travel times of plan i.

Then, instead of removing the plan with the lowest score, the plan i of a virtual person

[21] A leg for the air transport mode is already contained in the plan database if transit line and route are equal. Legs of the alternative mode are considered equal, if they connect the same activity locations by equal travel times. Note, that this implementation may be subject to change.

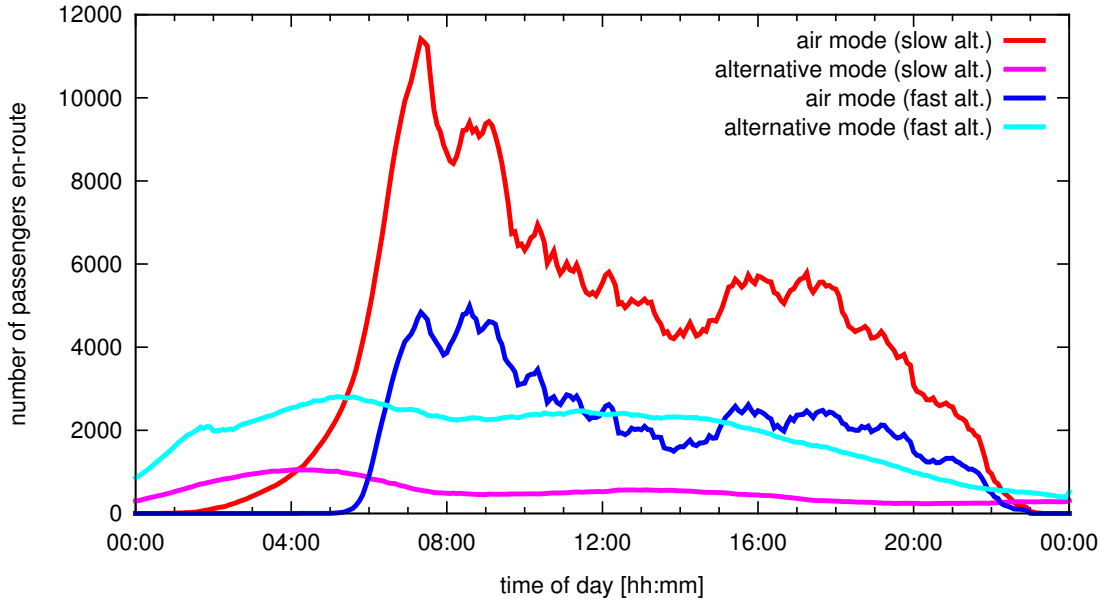


Figure 6.8.: Results with random selector for plan removal, iteration 600. Passengers waiting for a flight or traveling by plane or by the alternative mode over time of day

n for deletion is selected randomly with probability

$$p(i) \propto e^{-\mu(V_i+\beta_{PS}\ln PS_{in})},$$

where V_i is the score, μ the sensitivity parameter from Eq. 2.2, and β_{PS}, PS_{in} denote the same as above.

Results

The simulation runs are repeated with the same setup as for the runs that includes the alternative mode. Plans are deleted by the presented random selector, $\beta_{PS} = 60$. Fig. 6.8 shows the resulting travel patterns over time for alternative modes at speed $100\,km/h$ and $300\,km/h$. Travelers on the alternative mode are distributed more homogeneously over time of day. The speed increase of the alternative mode attracts more passengers. This is reflected by the modal splits in Tab. 6.5. Only one passenger gets stuck at the end of day. The mean square error is higher than without the alternative mode (Tab. 6.6). This is plausible as the data for the demand only contains air transport trips.

6.5. Discussion

Overall, the results show that a microscopic, agent-based simulation of passenger demand for air transport is feasible. Most passengers are able to learn the constraints of air transport technology and arrive at their desired destination.

$v[km/h]$	# air mode	# alt. mode	# stuck	air mode[%]	alt. mode[%]	stuck[%]
100	49280	2551	1	95.08	04.92	00.00
150	44835	6996	1	86.50	13.50	00.00
200	39929	11902	1	77.04	22.96	00.00
250	34332	17499	1	66.24	33.76	00.00
300	27270	24562	0	52.61	47.39	00.00

Table 6.5.: Results with random selector for plan removal, iteration 600. Modal split for different speeds of the alternative mode

$v[km/h]$	σ^2	σ	mean rel error	stuck
$od_{transfer} - od_{direct}$	12640	112	1.75	-
100	10367	102	0.35	1
150	13820	118	0.43	1
200	18651	137	0.56	1
250	25291	159	0.68	1
300	36059	190	0.76	0

Table 6.6.: Results with random selector for plan removal, iteration 600. Simulation results including an alternative mode at different speeds v

6.5.1. Air Transport Only

Stuck

Without the alternative mode, the only available transport mode is a capacity restricted flight connection that is served in discrete, irregular time intervals. The number of stuck passengers is higher than for the simulation runs with the alternative mode. Passengers get more likely stuck on O-D pairs where the demand excesses seat capacity. This may have model extrinsic and intrinsic reasons.

Extrinsic Stuck

Choice, quality, and preprocessing of available data sources is extrinsic. The quality of the simulation model's outcome hinges at the data available. For older versions of the air transport passenger demand, DESTATIS data for 09-2011 were used[22]. The air transport technology model, however, was created on a 09-2009 flight schedule. The number of starts of flights within Germany increased slightly between 2009 and 2011 (DLR, 2012, p. 23). Assuming that the number of available seats is increased accordingly, the simulation model provided too little capacity, at least on certain O-D pairs. As result, the number of passengers that had not reached their destination but got stuck was much higher. These results can be found in Appendix C. With the availability of 09-2009 DESTATIS data, the overall quality of results increased. The replacement of the 2011 data by 2009 data reduced the number of stuck passenger significantly, from

[22] For some reason, DESTATIS provides historical data up to 01-2010. Older data is not available. Special thanks to Dr. Tobias Grosche for providing the 2009 DESTATIS data.

around 1500 to 350 travelers.

Data is provided on a monthly basis, while the time horizon of the simulation model is one day. The number of trips per day is retrieved on the assumption that trips are uniformly distributed over all days of a month. The remaining 350 stuck passengers might be resolved by a more accurate distribution. Otherwise, a longer time horizon could be simulated[23]. This would also include flights that are not departing on a Tuesday. Possibly, travelers no longer get stuck.

Intrinsic Stuck

The problem of stuck passenger can be model-intrinsic. If there is only one, early, connection to a hub per day, departure times of some passengers might be too late to reach that connection. The random departure time mutation may not be able to find that connection for all passengers. This has been ruled out for the current setup but should be considered in further studies.

Alternatively, it may be the case that passengers have a connection that works in theory, but they are "crowded out" by other passengers who arrive earlier at the gate. They would make it if either of them would take a different route. The current approach would not find such a solution, since passengers do not take into account the costs they impose on others, see Lämmel and Flötteröd (2009) for an approach to take that into account. The real-world solution presumably would be to raise prices on congested seats until one or the other passenger re-routes. Currently, all passengers have homogeneous values of time. For a more meaningful price modeling, more heterogeneous attributes of passengers can be included. As the present model is based on sole O-D data, it does not include such a process. In principle, other data, as e.g. Lorenz curves and median incomes, can be merged with the O-D data (Kickhöfer et al., 2011).

Diversity Routing

An alternative approach to remove some of these shortcomings might be to use a router that generates a larger diversity of routes even for the same departure time. Such a router would be able to point a passenger to a route where seats are available without by itself knowing about seat availability. That approach would, however, not address the issue that some passengers might need to switch their path in order to allow *others* to obtain a feasible path. In Graf (2013), a first prototype of such a router is tested in a different context. First tests for the flight model revealed only slight improvements. As more diverse routes are dominated by the direct connection, they are removed by the algorithm similar to routes on slow alternative modes. After this more general problem is solved, a more diverse routing should be reconsidered.

6.5.2. Alternative Mode

Interpretation

The alternative mode can be interpreted as mixture between train, bus, or car connection availability. Clearly, the results hinge at the assumption that the alternative mode

[23] Note, that this requires some changes in the source code that may not be resolved by sole customizations of MATSim. Please ask the developers before running MATSim for a longer time horizon.

is always available and not capacity restricted. All passengers on the alternative mode face the same travel speed. This assumption is too coarse for the presented scenario. E.g., average speed and temporal availability of train connections depends on the O-D pair. In principle, the alternative mode could be refined by inclusion of O-D pair dependent average speed data. Alternatively, train, bus, and car can be simulated explicitly, featuring capacity restrictions and mutual interactions. For illustration of the overall modeling approach, however, a homogeneous velocity for the alternative mode seems to be more appropriate.

Effects

The effects evoked by the availability of the alternative mode are illustrative. The data for the demand provides O-D pairs for air transport, but not for car, train or bus trips. Thus, the simulation results should get worse when the alternative mode is added. This is reflected by the presented results. For more plausible interpretations, further data for demand on other modes is required. Alternatively, the 2011 data could be used.

6.5.3. Overall Approach

Extent

All modeling approaches explain the routing of passengers in more detail than it can be solely retrieved from the input data. The quantity of reaction, however, seems to be relatively small. Most passengers use a direct connection. Besides the issues discussed so far, this is highly plausible. Consider the geospatial extent of the demand. Flying within Germany is often not worth it, if the connection includes a transfer. Then, empirically it is faster to travel by train, car, or bus. To gain further insights, the geospatial extent of the modeled demand could be increased. Data for all Europe is still hard to collect. DESTATIS data, however, contains also O-D data for worldwide trips touching Germany. Adding these O-D pairs to the presented approach is conceptually straightforward. The number of connections that contain a transfer should increase. Also, seat occupancy gets a valid interpretation.

Time Structure

Passengers are modeled without explicit desired departure or arrival times. The simulation approach can capture such individual time constraints. Input data for this study, however, contains monthly O-D pairs without any further information about time distribution. A detailed modeling of individual time constraints is not considered in this study. With some more data, the information can be added without big effort.

Potential Applications

Clearly, potential applications of the proposed model depend on type and detail of included information. In general, application for policy planning allows a more detailed evaluation of the effects from mid-distance travel policies that includes consideration of mode alternatives. The approach could also be useful for private companies, planning flight-schedules and capacities on different connections. The impacts of changes on customers can be assessed on a high level of detail.

6.6. Findings

Technology

In the first part of this chapter, a microscopic modeling approach for air transport technology is presented. A multi-agent simulation, originally developed for urban transport planning and forecasting, is used. Aircraft are represented microscopically, featuring attributes as speed, available seats, and boarding constraints. The air traffic network and flight performance is captured at a low level of detail as air traffic management applications are not intended. Despite the lack of detail, some relevant aspects of congestion and delays can be captured by a queue model for traffic flow. The queue model is computationally relatively cheap so large scenarios can be simulated. As proof of example results for the Europe to world air transport are presented. Overall modeling of technology is similar to the approach by Clarke et al. (2007), the level of detail is, however, coarser. In principle, in the same way as Clarke et al. (2007), further models for, e.g., gates, taxiing, weather or airline operations could be added to the presented approach. In contrast, and going beyond Clarke et al. (2007), passengers are captured at all stages of their trip. Further, passengers traveling on alternative transport modes can be simulated.

Passengers

An agent-based modeling of passengers is subject of the second part of this chapter. For this, a passenger demand for the German national air transport system is set up. The computationally affordable simulation technique enables an iterative, simulation-based assignment of passengers to flights of the Europe to world wide model for air transport technology. Furthermore, alternative transport modes can be added. Overall, the presented results look promising. Some problems of the overall simulation technique are uncovered. These are more general and not specific to air transport systems. Several solutions are presented and discussed. Potentially, the overall problem can be solved with some of them. Further research, however, is required before the passenger model can be refined and calibrated. Then, the model may help to get a more detailed picture of mid-distance travel patterns.

Chapter 7

Conclusion

The thesis covers three areas: traffic signal control, air transport systems and software engineering. Observations are made from a software engineering perspective within the development process of a multi-agent transport simulation — MATSim.

Traffic Signal Control

Traffic signal control can be simulated with the multi-agent approach. Traffic flow at signalized intersections is simulated by a queue model (Gawron, 1998b; Simon et al., 1999; Cetin, 2005). This is the default traffic flow model of the simulation. Going beyond (Gawron, 1998b; Simon et al., 1999; Cetin, 2005), the thesis provides a detailed specification for the modeling of traffic signals. The traffic signal model and the underlying implementation are optional extensions to the overall simulation process. It can be employed without modifications of the transport network. The implementation can be extended or exchanged completely. Based on the traffic signal model, network wide effects on travel patterns are studied that result from a change of traffic signal control. Traffic patterns can be worse, if a fixed-time control is replaced by a traffic-responsive control. It is shown, that the simulation can capture these instabilities. First results from an ongoing project indicate, that different choice dimensions of travelers influence the evaluation of traffic signal control. An optimized coordination of fixed-time control appears quite stable for different choice dimensions and travel patterns. The overall approach is suited to study network wide effects of traffic-responsive and optimized traffic signal control.

Air Transport Systems

The multi-agent simulation can be applied to air transport systems. Therefore, it is shown, how technology of air transport systems can be modeled. Then, individual passengers are modeled on all stages of their trip including their choice of transport mode. Thus, competitive markets, e.g., between high speed rail and air transport, can be analyzed. Results reveal, however, a more general problem of the currently implemented mode choice model. A potential solution is presented, that may provide further insights and implications to improve the quality of the software.

Software Engineering

The software engineering part of this thesis explains backgrounds and implications

of the design decisions undertaken while the first 2007 prototype of MATSim was redesigned. The prototype was a monolithic piece of hard to customize software, and thus not suited for further research. The redesign targets at a modular, extensible architecture that permits researchers to modify or add certain components to the overall simulation. As proof of concept, the module for fixed-time traffic signal control is provided as extension. The module is completely decoupled from the overall simulation approach and can be replaced in part, or completely.

Research Software

Decoupled modules increase quality of software based research. They are more stable than monolithic software structures. The latter face the problem that each refactoring of some component implies changes to many other components. A change to a component may imply that code reviews and proofs of concept have to be repeated. Clearly, sole decoupling is not solving all problems of research software. Revisiting the argumentation in the motivation of this thesis, there is no general answer how to write and maintain research software. Further, there is no overall agreement to responsibilities for selection, testing, and technical documentation of software based research. Frequently, none of these steps is undertaken carefully. Thus, software based research may indeed be considered critically. Within a single work these problems obviously cannot be solved.

Contributions

Some contributions, however, can be found within this thesis and may support further considerations. Based on well established concepts from computer science and software engineering, it is argued that concerns should be separated. This applies to the separation of data, computational functions, and user iterfaces, as well as to the use of standards instead of customized solutions. Code (peer) reviews are eased by such architectures. A reviewer can focus on the part of the software that is subject to the review.

Open Source

The code is open source, version controled, and hosted on a publicly accessible server. The build sequence of the software is, besides abuse, reproduceable. In conjunction with the information provided in Appendix A, the presented simulation runs can be reproduced. Within the open source software development process, branching is frequently used. Branching is considered helpful for the development of new software components. For peer reviewing software functionality, however, branching is cumbersome. The complexity to understand changes made to a branch increases quickly. The review and all studies must be repeated, when the branch is merged with the main development code. Therefore, the software architecture described in this thesis encourages customizable code. The responsiblity to motivate and describe such customizations accurately has to be beared by the researcher. Journals, that publish the research results, may encourage researchers to provide such documentation.

Development Cycle

The results for the air transport model presented in Chapter 6 nicely illustrate the development cycle of the simulation. A simulation model, that provides plausible answers to partial problems, can be programmed much faster than it can be made consistent with theory. A lack of resources for more theoretical considerations may explain this

practice, but also, a lack of generally accepted or applicable theory. The latter may motivate further theoretical work that is eased by the white box simulation. Awareness of this development cycle is considered essential for researches. Before certain features of the simulation are selected for further studies, some tests are essential. Clearly, further improvements can be undertaken to support software selection and use.

Further Steps

Peer reviewed software may support the selection process of researchers. But, as long as software is under development, there is a strong incentive to use more recent versions of a software than the peer reviewed version. Functionality, that worked in the peer reviewed version, can be faulty in a newer version. Further, sole peer reviewing might not uncover all effects that emerge from the implemented algorithmic. These deficiencies could be remedied in part by a certain set of default scenarios. For certain builds of a software, these scenarios can be analyzed automatically. The outcome of analysis then could be provided to researchers, e.g., as pdf file, and may help to select a certain software. Further, it may help reviewers to assess changes made to default functionality. Such scenarios can be set up on artifical instances that allow theoretical calculations (e.g. Rieser et al., 2009), and on real-world instances. The latter requires publicly available data, that is still hard to gather for the transportation domain. In the last years, however, situation improved significantly in some areas, e.g., by the availablity of openstreetmap data. Currently, a lot of effort is undertaken for open access to public transit schedules.

Conclusion

In this thesis, several big fields of transport engineering are considered. Each of them could, indeed, be adressed in more detail. Specialization, however, impedes a broader view on the multi-agent simulation, that seems to be required for maintenance. This consideration may justify the chosen approach. Therefore, at some points in this work, we stop under the assumption that someone can take over who is more specialized in the field, fully aware that we don't know what we don't know.

Bibliography

K. Aboudolas, M. Papageorgiou, A. Kouvelas, and E. Kosmatopoulos. A rolling-horizon quadratic-programming approach to the signal control problem in large-scale congested urban road networks. *Transportation Research Part C: Emerging Technologies*, 18(5):680–694, 2010. ISSN 0968-090X. doi: DOI:10.1016/j.trc.2009.06.003. Applications of Advanced Technologies in Transportation: Selected papers from the 10th AATT Conference.

A. Agarwal, M. Zilske, K.R. Rao, and K. Nagel. Person-based dynamic traffic assignment for mixed traffic conditions. In *Conference on Agent-Based Modeling in Transportation Planning and Operations 2013*, Blacksburg, Virginia, USA, 2013. Also VSP WP 12-11, see `www.vsp.tu-berlin.de/publications`.

S. Alam, H.A. Abbass, and M. Barlow. Atoms: Air traffic operations and management simulator. *Intelligent Transportation Systems, IEEE Transactions on*, 9(2):209–225, 06 2008. ISSN 1524-9050. doi: 10.1109/TITS.2008.922877.

R.E. Allsop. Estimating the traffic capacity of a signalized road junction. *Transportation Research*, 6(3):245–255, 1972. ISSN 0041-1647. doi: 10.1016/0041-1647(72)90017-2.

R.E. Allsop. Signal control at individual junctions: Stage-based approach. In M. Papageorgiou, editor, *Concise Encyclopedia of Traffic & Transportation Systems*, Advances in systems, control and information engineering, pages 478–483. Pergamon Press, Oxford, 1. edition, 1991. ISBN 0-08-036203-6.

M. Balmer. *Travel demand modeling for multi-agent transport simulations: Algorithms and systems*. PhD thesis, Swiss Federal Institute of Technology (ETH) Zürich, Switzerland, 2007.

M. Balmer, M. Bernard, and K.W. Axhausen. Matching geo-coded graphs. In *Proceedings of the Swiss Transport Research Conference (STRC)*, Monte Verita, Switzerland, 2005a. URL `http://www.strc.ch`.

M. Balmer, B. Raney, and K. Nagel. Adjustment of activity timing and duration in an agent-based traffic flow simulation. In H.J.P. Timmermans, editor, *Progress in activity-based analysis*, pages 91–114. Elsevier, Oxford, UK, 2005b.

J. Barceló, E. Codina, J. Casas, J. L. Ferrer, and D. García. Microscopic traffic simulation: A tool for the design, analysis and evaluation of intelligent transport systems. *Journal of Intelligent & Robotic Systems*, 41:173–203, 2005. ISSN 0921-0296. doi: 10.1007/s10846-005-3808-2.

A.L.C. Bazzan. A distributed approach for coordination of traffic signal agents. *Autonomous Agents and Multiagent Systems*, 10(1):131–164, March 2005.

A.L.C. Bazzan. Opportunities for multiagent systems and multiagent reinforcement learning in traffic control. *Autonomous Agents and Multi-Agent Systems*, 18:342–375, 2009. ISSN 1387-2532. doi: 10.1007/s10458-008-9062-9.

A.L.C. Bazzan, D. de Oliveira, F. Klügl, and K. Nagel. Adapt or not adapt – consequences of adapting driver and traffic light agents. In K. Tuyls, A. Nowe, Z. Guessoum, and D. Kudenko, editors, *Adaptive Agents and Multi-Agent Systems III. Adaptation and Multi-Agent Learning*, volume 4865 of *Lecture Notes in Computer Science*, pages 1–14. Springer, Berlin/Heidelberg, 2008. ISBN 978-3-540-77947-6. doi: 10.1007/978-3-540-77949-0_1.

A.L.C. Bazzan, M. de B. do Amarante, T. Sommer, and A. J. Benavides. ITSUMO: An agent-based simulator for ITS applications. In Rosaldo Rossetti, Henry Liu, and Shuming Tang, editors, *Proceedings of the 4th Workshop on Artificial Transportation Systems and Simulation*. IEEE, September 2010a.

A.L.C. Bazzan, D. de Oliveira, and B. C. da Silva. Learning in groups of traffic signals. *Engineering Applications of Artificial Intelligence*, 23:560–568, June 2010b. ISSN 0952-1976. doi: http://dx.doi.org/10.1016/j.engappai.2009.11.009.

A.L.C. Bazzan, M. de B. do Amarante, G. G. Azzi, A. J. Benavides, L. S. Buriol, L. Moura, M. P. Ritt, and T. Sommer. Extending traffic simulation based on cellular automata: from particles to autonomous agents. In Tadeusz Burczynski, Joanna Kolodziej, Aleksander Byrski, and Marco Carvalho, editors, *Proceedings of the Agent-Based Simulation (ABS / ECMS 2011)*, pages 91–97, Krakow, June 2011. ECMS.

M. C. Bell and R. D. Bretherton. Ageing of fixed-time traffic signal plans. In *2nd International Conference on Road Traffic Control*, London, April 1986. IEE.

M. Ben-Akiva and S. R. Lerman. *Discrete choice analysis*. The MIT Press, Cambridge, MA, 1985.

C. Bielefeldt and F. Busch. MOTION – A new on-line traffic signal network control system. In *Seventh International Conference on Road Traffic Monitoring and Control*, pages 55 –59, 04 1994. doi: 10.1049/cp:19940424.

K. Bilimoria, B. Sridhar, G. Chatterji, K. Sheth, and S. Grabbe. FACET: Future ATM Concepts Evaluation Tool. 3rd USA/Europe Air Traffic Management R & D Seminar, Napoli, Italy, 06 2000.

J. Birkmann, S. Dech, N. Goseberg, H. Klüpfel, G. Lämmel, F. Moder, K. Nagel, M. Oczipka, T. Schlurmann, N. Setiadi, F. Siegert, G. Strunz, and H. Taubenböck. Numerical last-mile tsunami early warning and evacation information system ("Last-Mile – Evacuation "). In *GEOTECHNOLOGIEN Science Report No. 10: "Early Warning Systems in Earth Management"*, pages 73–83, Osnabrück, October 2008.

J. Bischoff. Verkehrsabhängige Lichtsignalanlagensteuerung – Vergleich und simulationsbasierte Evaluation. Bachelor's thesis, TU Berlin, Institute for Land and Sea Transport Systems, Berlin, Germany, December 2010.

T. Böhme. Analyse und Simulation von Verspätungen im europäischen Linien-Luftverkehr. Bachelor's thesis, TU Berlin, Institute for Land and Sea Transport Systems, Berlin, Germany, October 2011.

D. Braess. Über ein Paradoxon aus der Verkehrsplanung. *Unternehmensforschung*, 12: 258–268, 1968.

R. Braun, F. Busch, C. Kemper, R. Hildebrandt, F. Weichenmeier, C. Menig, I. Paulus, and R. Preßlein-Lehle. TRAVOLUTION – Netzweite Optimierung der Lichtsignalsteuerung und LSA-Fahrzeug-Kommunikation. *Straßenverkehrstechnik*, 06:365–374, 2009.

D. Bretherton, M. Bodger, and N. Baber. SCOOT–The future. In *Road Transport Information and Control, 2004. RTIC 2004. 12th IEE International Conference on*, pages 301–306, 04 2004. doi: 10.1049/cp:20040045.

W. Brilon, T. Wietholt, A. Pott, and Zelke U. Adaptive koordinierte Signalsteuerung in Münster. *Straßenverkehrstechnik*, 9:565–573, 2009.

B. Bubalo and J. Daduna. Airport capacity and demand calculations by simulation–the case of Berlin-Brandenburg International Airport. *NETNOMICS*, pages 1–21, 2012. ISSN 1385-9587. doi: 10.1007/s11066-011-9065-6.

W. Burghout and J. Wahlstedt. Hybrid traffic simulation with adaptive signal control. *Transportation Research Record*, 1999:191–197, 2007. doi: http://dx.doi.org/10.3141/1999-20.

F. Busch and G. Kruse. MOTION for SITRAFFIC - a modern approach to Urban Traffic Control. In *Intelligent Transportation Systems, 2001. Proceedings. 2001 IEEE*, pages 61 –64, 2001. doi: 10.1109/ITSC.2001.948630.

J. Carver, D. Heaton, L. Hochstein, and R. Bartlett. Self-perceptions about software engineering: A survey of scientists and engineers. *Computing in Science Engineering*, 15(1):7–11, 2013. ISSN 1521-9615. doi: 10.1109/MCSE.2013.12.

E. Cascetta, M. Gallo, and B. Montella. Models and algorithms for the optimization of signal settings on urban networks with stochastic assignment models. *Annals of Operations Research*, 144:301–328, 2006. ISSN 0254-5330. doi: 10.1007/s10479-006-0008-9.

Central Office for Delay Analysis. CODA Digest – Delays to Air Transport in Europe Annual 2011. Technical report, Eurocontrol, 03 2012.

N. Cetin. *Large-Scale parallel graph-based simulations*. PhD thesis, Swiss Federal Institute of Technology (ETH) Zürich, Switzerland, 2005.

N. Cetin, A. Burri, and K. Nagel. A large-scale agent-based traffic microsimulation based on queue model. In *Proceedings of the Swiss Transport Research Conference (STRC)*, Monte Verita, Switzerland, 2003. URL `http://www.strc.ch`. Earlier version, with inferior performance values: Transportation Research Board Annual Meeting 2003 paper number 03-4272.

D. Charypar. *Efficient algorithms for the microsimulation of travel behavior in very large scenarios*. PhD thesis, Swiss Federal Institute of Technology (ETH) Zürich, Switzerland, 2008.

D. Charypar and K. Nagel. Generating complete all-day activity plans with genetic algorithms. *Transportation*, 32(4):369–397, 2005. ISSN 0049-4488. doi: 10.1007/s11116-004-8287-y.

O.J. Chen and M.E. Ben-Akiva. Game-theoretic formulations of interaction between dynamic traffic control and dynamic traffic assignment. *Transportation Research Record*, 1617:179–188, 1998.

Y. Chen, M. Rieser, D. Grether, and K. Nagel. Improving a large-scale agent-based simulation scenario. VSP Working Paper 08-15, TU Berlin, Transport Systems Planning and Transport Telematics, 2008. See `www.vsp.tu-berlin.de/publications`.

C. Christina Diakaki, V. Vaya Dinopoulou, K. Kostas Aboudolas, M. Markos Papageorgiou, E. Ben-Shabat, E. Seider, and A. Leibov. Extensions and New Applications of the Traffic-Responsive Urban Control Strategy: Coordinated Signal Control for Urban Networks. *Transportation Research Record: Journal of the Transportation Research Board*, 1856(1):202–211, 01 2003. doi: 10.3141/1856-22.

J.-P. Clarke, T. Melconian, E. Bly, and F. Rabbani. MEANS–MIT extensible air network simulation. *Simulation*, 83(5):385–399, 05 2007. ISSN 0037-5497. doi: 10.1177/0037549707063766.

European Commission. Airport policy in the European Union – addressing capacity and quality to promote growth, connectivity and sustainable mobility. COM(2011) 823 final, 12 2011.

S. B. Cools, C. Gershenson, and B. D'Hooghe. Self-organizing traffic lights: A realistic simulation. In Mikhail Prokopenko, editor, *Self-Organization: Applied Multi-Agent Systems*, chapter 3, pages 41–49. Springer, 2007.

National Research Council, editor. *Highway Capacity Manual 2000 (Metric Version)*. TRB, 2000. ISBN 0-309-06681-6.

M. Cremer and M. Landenfeld. A mesoscopic model for saturated urban road networks. In M. Schreckenberg and D.E. Wolf, editors, *Traffic and Granular Flow '97*, pages 169 –180. Springer, Singapore, 1998.

C.F. Daganzo. Queue spillovers in transportation networks with a route choice. *Transportation Science*, 32(1):3–11, 1998.

C.F. Daganzo. Urban gridlock: Macroscopic modeling and mitigation approaches. *Transportation Research Part B: Methodological*, 41(1):49–62, 2007. ISSN 0191-2615. doi: 10.1016/j.trb.2006.03.001.

C.F. Daganzo and N. Geroliminis. An analytical approximation for the macroscopic fundamental diagram of urban traffic. *Transportation Research Part B: Methodological*, 42(9):771–781, 2008. ISSN 0191-2615. doi: 10.1016/j.trb.2008.06.008.

D. De Roure and C. Goble. Software design for empowering scientists. *Software, IEEE*, 26(1):88–95, 2009. ISSN 0740-7459. doi: 10.1109/MS.2009.22.

C. Diakaki, M. Papageorgiou, and K. Aboudolas. A multivariable regulator approach to traffic-responsive network-wide signal control. *Control Engineering Practice*, 10(2): 183–195, 2002. ISSN 0967-0661. doi: 10.1016/S0967-0661(01)00121-6.

R. Diestel. *Graph Theory*. Springer-Verlag, Heidelberg, 4th edition, corrected reprint 2012 edition, 2010. ISBN 978-3-642-14278-9.

E. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1:269–271, 1959.

DLR. Luftverkehrsbericht 2011 – Daten und Kommentierungen des deutschen und weltweiten Luftverkehrs. Technical report, Deutsches Zentrum für Luft- und Raumfahrt e.V. – DLR, 2012.

W. Elmenreich, R. D'Souza, C. Bettstetter, and H. de Meer. A survey of models and design methods for self-organizing networked systems. In T. Spyropoulos and K. Hummel, editors, *Self-Organizing Systems*, volume 5918 of *Lecture Notes in Computer Science*, pages 37–49. Springer Berlin / Heidelberg, 2009. ISBN 978-3-642-10864-8. URL `http://dx.doi.org/10.1007/978-3-642-10865-5_4`.

European Environment Agency. Corine Land Cover 2006 seamless vector data, 08 2011. URL `www.eea.europa.eu/data-and-maps/data/ds_resolveuid/f7d1b28c05ce9810df31852becd1c645`. version 15.

C.S. Fisk. Game theory and transportation systems modelling. *Transportation Research Part B: Methodological*, 18(4-5):301–313, 1984. ISSN 0191-2615. doi: 10.1016/0191-2615(84)90013-4.

G. Flötteröd and G. Lämmel. Evacuation simulation with limited capacity sinks – an evolutionary approach to solve the shelter allocation and capacity assignment problem in a multi-agent evacuation simulation. In *Proceedings of the International Conference on Evolutionary Computation*, pages 249–254. SciTePress, 2010. ISBN 978-989-8425-31-7.

L.R. Ford and D.R. Fulkerson. *Flows in Networks*. Princeton University Press, 1962.

M. Fowler. Reducing coupling. *Software, IEEE*, 18(4):102–104, 07–08 2001. ISSN 0740-7459. doi: 10.1109/MS.2001.936226.

M. Fowler. Inversion of control containers and the dependency injection pattern. online article, 01 2004a. URL `http://martinfowler.com/articles/injection.html`.

M. Fowler. *Refactoring: Improving the design of existing code*. Addison-Wesley, 2004b.

E. Frejinger and M. Bierlaire. Capturing correlation with subnetworks in route choice models. *Transportation Research Part B: Methodological*, 41(3):363 – 378, 2007. ISSN 0191-2615. doi: 10.1016/j.trb.2006.06.003.

B. Friedrich. Verkehrsadaptive Steuerung von Lichtsignalanlagen - Ein Überblick. Festschrift zum Ehrenkolloquium für Univ.- Prof. Dr./UCB Hartmut Keller 01.03.2002 an der TU München, Veröffentlichungen des Fachgebiets Verkehrstechnik und Verkehrsplanung der Technischen Universität München, München, 2002.

S. Fürbas. Modellierung und Simulation des europäischen Linien–Luftverkehrs. Bachelor's thesis, TU Berlin, Institute for Land and Sea Transport Systems, Berlin, Germany, 03 2011.

E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns. Elements of reusable object-oriented software*. Addison-Wesley, 1995.

N. Gartner and C. Stamatiadis. Integration of dynamic traffic assignment with real-time traffic adaptive control system. *Transportation Research Record: Journal of the Transportation Research Board*, 1644(1998):150–156, 2007. doi: 10.3141/1644-16.

N. H. Gartner and P. Wagner. Analysis of traffic flow characteristics on signalized arterials. *Transportat Research Record: Journal of the Transportation Research Board*, 1883: 94–100, 2004. doi: 10.3141/1883-11.

C. Gawron. An iterative algorithm to determine the dynamic user equilibrium in a traffic simulation model. *International Journal of Modern Physics C*, 9(3):393–407, 1998a.

C. Gawron. *Simulation-based traffic assignment*. PhD thesis, University of Cologne, Cologne, Germany, 1998b.

R. Geisberger. *Advanced Route Planning in Transportation Networks*. PhD thesis, Karsruhe Institut für Technologie, 2011.

B. George and S. Shekhar. Time-aggregated graphs for modeling spatio-temporal networks. In S. Spaccapietra, J.Z.. Pan, P. Thiran, T. Halpin, S. Staab, V. Svatek, P. Shvaiko, and J. Roddick, editors, *Journal on Data Semantics XI*, volume 5383 of *Lecture Notes in Computer Science*, pages 191–212. Springer Berlin Heidelberg, 2008. ISBN 978-3-540-92147-9. doi: 10.1007/978-3-540-92148-6_7.

N. Geroliminis and C. F. Daganzo. Existence of urban-scale macroscopic fundamental diagrams: Some experimental findings. *Transportation Research Part B*, 42:759–770, 2008.

C. Gershenson and D.A. Rosenblueth. Modeling self-organizing traffc lights with elementary cellular automata. Technical report, Instituto de Investigaciones en Matematicas Aplicadas y en Sistemas Universidad Nacional Autonoma de Mexico, 2009.

GEVAS software Systementwicklung und Verkehrsinformatik GmbH. BALANCE – intelligente verkehrsadaptive Netzsteuerung. online, 2011. URL `www.gevas.eu/download/`.

P. G. Gipps. A behavioural car-following model for computer simulation. *Transportation Research B*, 15:105–111, 1981.

A. Graf. Die Bewertung der Qualität des Schülerverkehrs unter Anwedung der Multiagentensimulation MATSim. Diplomarbeit (Diploma Thesis), TU Berlin, Institute for Land and Sea Transport Systems, Berlin, Germany, 05 2013.

D. Grether and K. Nagel. Agent-based modelling of air transport demand. In *Presented at the 2013 World Conference of Air Transport Research Society*, Bergamo, Italy, June 2013a.

D. Grether and K. Nagel. Extensible software design of a multi-agent transport simulation. *Procedia Computer Science*, 19:380–388, 2013b. ISSN 1877-0509. doi: 10.1016/j.procs.2013.06.052. URL `http://www.sciencedirect.com/science/article/pii/S1877050913006601`.

D. Grether, M. Rieser, and K. Nagel. Transport systems' planning and transport telematics at a quick glance. Presentation, 07 2007. URL `http://www.matsim.org/node/107`.

D. Grether, Y. Chen, M. Rieser, U. Beuck, and K. Nagel. Emergent effects in multi-agent simulations of road pricing. In *Proceedings of the Annual Meeting of the European Regional Science Association (ERSA)*, 2008. Also VSP WP 08-08, see `www.vsp.tu-berlin.de/publications`.

D. Grether, Y. Chen, M. Rieser, and K. Nagel. Effects of a simple mode choice model in a large-scale agent-based transport simulation. In A. Reggiani and P. Nijkamp, editors, *Complexity and Spatial Networks. In Search of Simplicity*, Advances in Spatial Science, chapter 13, pages 167–186. Springer, 2009a. doi: 10.1007/978-3-642-01554-0.

D. Grether, B. Kickhöfer, and K. Nagel. Policy evaluation in multi-agent transport simulations considering income-dependent user preferences. In *Proceedings of The 12th Conference of the International Association for Travel Behaviour Research (IATBR)*, Jaipur, India, 2009b. URL `www.iatbr.org`. Also VSP WP 09-13, see `www.vsp.tu-berlin.de/publications`.

D. Grether, J. Bischoff, and K. Nagel. Traffic-actuated signal control: Simulation of the user benefits in a big event real-world scenario. In *2nd International Conference on Models and Technologies for ITS, Leuven, Belgium*, 2011.

D. Grether, A. Neumann, and K. Nagel. Simulation of urban traffic control: A queue model approach. *Procedia Computer Science*, 10:808–814, 2012. doi: 10.1016/j.procs.2012.06.104.

D. Grether, S. Fürbas, and K. Nagel. Agent-based modelling and simulation of air transport technology. *Procedia Computer Science*, 2013.

M. Hansen, T. Nikoleris, D. Lovell, K. Vlachou, and A. Odoni. Use of Queuing Models to Estimate Delay Savings from 4D Trajectory Precision. In *Proceedings 8th USA/Europe Seminar on Air Traffic Management Research & Development*, 2009.

L. Hatton. The chimera of software quality. *Computer*, 40(8):104–103, 2007. ISSN 0018-9162. doi: 10.1109/MC.2007.292.

C. Heinlein. Vertical, horizontal, and behavioural extensibility of software systems. Ulmer Informatik-Berichte 2003-06, Fakultät für Informatik, Universität Ulm, 2003. URL `http://vts.uni-ulm.de/docs/2005/5344/vts_5344.pdf`.

C. Heinlein. Open types and bidirectional relationships as an alternative to classes and inheritance. *Journal of Object Technology*, 6(3):101–151, 03 2007. ISSN 1660-1769. doi: 10.5381/jot.2007.6.3.a3.

J. Hofbauer and K. Sigmund. *Evolutionary games and replicator dynamics*. Cambridge University Press, 1998.

Y. Hollander and J. Prashker. The applicability of non-cooperative game theory in transport analysis. *Transportation*, 33:481–496, 2006. ISSN 0049-4488. doi: 10.1007/s11116-006-0009-1.

A. Horni, K. Nagel, and K. Axhausen. High-resolution destination choice in agent-based models. Annual Meeting Preprint 12-1988, Transportation Research Board, Washington, D.C., 2012. Also VSP WP 11-17, see `www.vsp.tu-berlin.de/publications`.

T.-Y. Hu and H.S. Mahmassani. Day-to-day evolution of network flows under real-time information and reactive signal control. *Transportation Research Part C*, 5(1):51–69, 1997. ISSN 0968-090X. doi: 10.1016/S0968-090X(96)00026-5.

P. B. Hunt, D. I. Robertson, R. D. Bretherton, and R. I. Winton. SCOOT– A Traffic Responsive Method of Coordinating Signals. TRRL Laboratory Report 1014, TRRL, Crowthorne, Berkshire, UK, 1981.

ITP/BVU. Prognose der deutschlandweiten Verkehrsverflechtungen 2025. Technical report, 2005. URL `http://daten.clearingstelle-verkehr.de/220/`.

R. R. Jacob, M. V. Marathe, and K. Nagel. A computational study of routing algorithms for realistic transportation networks. *ACM Journal of Experimental Algorithms*, 4(1999es, Article No. 6), 1999. doi: 10.1145/347792.347814.

R. Johnson, J. Hoeller, K. Donald, C. Sampaleanu, R. Harrop, T. Risberg, A. Arendsen, D. Davison, D. Kopylenko, M. Pollack, T. Templier, E. Vervaet, P. Tung, B. Hale, A. Colyer, J. Lewis, C. Leau, M. Fisher, S. Brannen, R. Laddad, A. Poutsma, C. Beams, T. Abedrabbo, A. Clement, D. Syer, O. Gierke, R. Stoyanchev, and P. Webb. *Spring Framework Reference Documentation*, 3.2.1.release edition, 2013. URL `http://www.springsource.org/spring-framework#documentation`.

L.N. Joppa, G. McInerny, R. Harper, L. Salido, K. Takeda, K. O'Hara, D. Gavaghan, and S. Emmott. Troubling trends in scientific software use. *Science*, 340(6134):814–815, 2013. doi: 10.1126/science.1231535.

I. Kaddoura, B. Kickhöfer, A. Neumann, and A. Tirachini. Agent-based optimisation of public transport supply and pricing: Impacts of activity scheduling decisions and simulation randomness. *Transportation*, submitted.

D. Kelly, D. Hook, and R. Sanders. Five recommended practices for computational scientists who write software. *Computing in Science Engineering*, 11(5):48–53, 2009. ISSN 1521-9615. doi: 10.1109/MCSE.2009.139.

D.F. Kelly. A software chasm: Software engineering and scientific computing. *Software, IEEE*, 24(6):120–119, 2007. ISSN 0740-7459. doi: 10.1109/MS.2007.155.

B. Kickhöfer, D. Grether, and K. Nagel. Income-contingent user preferences in policy evaluation: application and discussion based on multi-agent transport simulations. *Transportation*, 38:849–870, 2011. ISSN 0049-4488. doi: 10.1007/s11116-011-9357-6.

E. Köhler and M. Strehler. Traffic signal optimization using cyclically expanded networks. In Thomas Erlebach and Marco Lübbecke, editors, *Proceedings of the 10th Workshop on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems*, number 14 in OpenAccess Series in Informatics (OASIcs), pages 114–129, Dagstuhl, Germany, 2010. Schloss Dagstuhl - Leibniz-Zentrum für Informatik. ISBN 978-3-939897-20-0. doi: http://dx.doi.org/10.4230/OASIcs.ATMOS.2010.114. URL `http://drops.dagstuhl.de/opus/volltexte/2010/2754`.

E. Köhler and M. Strehler. Signalanlagenoptimierung mit zyklisch expandierten Netzwerken. In *Proceedings of HEUREKA 11*, pages 314–333, Köln, 2011. FGSV Verlag.

E. Köhler, R.H. Möhring, and M. Skutella. Algorithmics of large and complex networks. chapter Traffic Networks and Flows over Time, pages 166–196. Springer-Verlag, Berlin, Heidelberg, 2009. ISBN 978-3-642-02093-3. doi: 10.1007/978-3-642-02094-0_9.

A. Kouvelas, K. Aboudolas, M. Papageorgiou, and E. B. Kosmatopoulos. A hybrid strategy for real-time traffic signal control of urban road networks. *Intelligent Transportation Systems, IEEE Transactions on*, PP(99):1–11, 2011. ISSN 1524-9050. doi: 10.1109/TITS.2011.2116156.

D. Krajzewicz, E. Brockfeld, J. Mikat, J. Ringel, C. Rössel, W. Tuchscheerer, P. Wagner, and R. Woesler. Simulation of modern Traffic Lights Control Systems using the open source Traffic Simulation SUMO. In J. Krüger, A. Lisounkin, and G. Schreck, editors, *Proceedings of the 3rd Industrial Simulation Conference 2005*, 2005.

W. Kraus, F.A. de Souza, R.C. Carlson, M. Papageorgiou, L.D. Dantas, E. Camponogara, E. Kosmatopoulos, and K. Aboudolas. Cost effective real-time traffic signal control using the TUC strategy. *Intelligent Transportation Systems Magazine, IEEE*, 2(4):6–17, winter 2010. ISSN 1939-1390. doi: 10.1109/MITS.2010.939916.

J. Krimmling. Sytem SYLVIA; Verkehrstechnische Beschreibung der Steuerverfahren, 12 1995. monograph type: technical report; department: ANL-SV.

P. Krüger. Verspätungen im Flugverkehr: Modellerweiterung für die Simulation des europäischen Linien-Luftverkehrs. Studienarbeit, TU Berlin, Institute for Land and Sea Transport Systems, Berlin, Germany, 07 2012.

G. Lämmel and G. Flötteröd. Towards system optimum: Finding optimal routing strategies in time-tependent networks for large-scale evacuation problems. volume 5803 of *LNCS (LNAI)*, pages 532–539, Berlin Heidelberg, 2009. Springer. doi: 10.1007/978-3-642-04617-9_67.

G. Lämmel, D. Grether, and K. Nagel. The representation and implementation of time-dependent inundation in large-scale microscopic evacuation simulations. *Transportation Research Part C: Emerging Technologies*, 18(1):84–98, 2010. ISSN 0968-090X. doi: DOI:10.1016/j.trc.2009.04.020.

S. Lämmer. *Reglerentwurf zur dezentralen Online-Steuerung von Lichtsignalanlagen in Straßennetzwerken*. PhD thesis, Dresden Institute of Technology, 2007.

S. Lämmer and D. Helbing. Self-control of traffic lights and vehicle flows in urban road networks. *Journal of Statistical Mechanics: Theory and Experiment*, 2008(04):04–019, 2008. doi: 10.1088/1742-5468/2008/04/P04019.

S. Lämmer and D. Helbing. Self-stabilizing decentralized signal control of realistic, saturated network traffic. Santa Fe Working Paper Nr. 10-09-019, 2010.

S. Lämmer and M. Treiber. Self-healing networks – gridlock prevention with capacity regulating traffic lights. In *Workshop on Technologies for the Organisation, Adaptation and Simulation of Transportation Systems. Sixth IEEE International Conference on Self-Adaptive and Self-Organizing Systems*, 2012.

N. Lefebvre and M. Balmer. Fast shortest path computation in time-dependent traffic networks. In *Proceedings of the Swiss Transport Research Conference (STRC)*, Monte Verita, Switzerland, 2007. URL `http://www.strc.ch`.

G. Lulli and A. Odoni. The European Air Traffic Flow Management Problem. *Transportation Science*, 41(4):431–443, 2007. doi: 10.1287/trsc.1070.0214.

R.C. Martin. *Clean code. A Handbook of Agile Software craftsmanship*. Pearson Education, Inc., Right and Contracts Department, 501 Boylston Street, Suite 900, Boston, MA 002116, USA, 2009.

D. McDermott. Artificial intelligence meets natural stupidity. *SIGART Bull.*, (57):4–9, April 1976. ISSN 0163-5719. doi: 10.1145/1045339.1045340.

C. Meneguzzer. Review of models combining traffic assignment and signal control. *Journal of Transportation Engineering*, 123(2):148–155, March/April 1997. doi: 10.1061/(ASCE)0733-947X(1997)123:2(148).

S.J. Metsker and W.C. Wake. *Design Patterns in Java*. Pearson Education, 2006.

K. Nagel. *Multi-agent transportation simulation (book in progress)*. 2005. See `www.vsp.tu-berlin.de/publications`.

K. Nagel and G. Flötteröd. Agent-based traffic assignment: Going from trips to behavioural travelers. In R.M. Pendyala and C.R. Bhat, editors, *Travel Behaviour Research in an Evolving World – Selected papers from the 12th international conference on travel behaviour research*, chapter 12, pages 261–294. International Association for Travel Behaviour Research, 2012. ISBN 978-1-105-47378-4.

K. Nagel and M. Schreckenberg. A cellular automaton model for freeway traffic. *Journal de Physique I France*, 2:2221–2229, 1992.

K. Nagel, M. Strauss, and M. Shubik. The importance of timescales: Simple models for economic markets. *Physica A*, 340(4):668–677, 2004.

K. Nagel, D. Grether, U. Beuck, Y. Chen, M. Rieser, and K.W. Axhausen. *Multi-agent transport simulations and economic evaluation*, volume 228 of *Journal of Economics and Statistics (Jahrbücher für Nationalökonomie und Statistik)*, pages 173–194. 2008.

A. Neumann. Modellierung und Evaluation von Lichtsignalanlagen in Queue-Simulationen. Diplomarbeit (Diploma Thesis), TU Berlin, Institute for Land and Sea Transport Systems, Berlin, Germany, 2008.

A. Neumann and K. Nagel. Avoiding bus bunching phenomena from spreading: A dynamic approach using a multi-agent simulation framework. VSP Working Paper 10-08, TU Berlin, Transport Systems Planning and Transport Telematics, 2010. URL `https://svn.vsp.tu-berlin.de/repos/public-svn/publications/vspwp/2010/10-08`. See `www.vsp.tu-berlin.de/publications`.

A. Neumann, M. Balmer, and M. Rieser. Converting a static trip-based model into a dynamic activity-based model to analyze public transport demand in Berlin. In M.J. Roorda and E.J. Miller, editors, *Travel Behaviour Research: Current Foundations, Future Prospects*, chapter 7, pages 151–176. International Association for Travel Behaviour Research (IATBR), 02 2014. ISBN 9781304715173.

T. W. Nicolai, L. Wang, K. Nagel, and P. Waddell. Coupling an urban simulation model with a travel model – A first sensitivity test. In *Computers in Urban Planning and Urban Management (CUPUM)*, Lake Louise, Canada, 2011. Also VSP WP 11-07, see `www.vsp.tu-berlin.de/publications`.

T. Nikoleris and M. Hansen. Queueing models for trajectory-based aircraft operations. *Transportation Science*, 2012. doi: 10.1287/trsc.1120.0411.

R. Oertel and P. Wagner. Delay-time actuated traffic signal control for an isolated intersection. Annual Meeting Preprint 11-1719, Transportation Research Board, Washington D.C., 2011.

J. Offutt. Putting the engineering into software engineering education. *Software, IEEE*, 30(1):96–94, 2013. ISSN 0740-7459. doi: 10.1109/MS.2013.12.

S. Pallottino and M.G. Scutella. Shortest path algorithms in transportation models: Classical and innovative aspects. In P. Marcotte and S. Nguyen, editors, *Equillibrium and Adavanced Transportation Modelling*, pages 245–281. Kluwer, 1998.

C. Petrie. No science without semantics. *Internet Computing, IEEE*, 11(4):88–87, 2007. ISSN 1089-7801. doi: 10.1109/MIC.2007.89.

H. Prothmann, F. Rochner, S. Tomforde, Branke. K, C. Müller-Schloer, and H. Schmeck. Organic Control of Traffic Lights. In *Proceedings of the 5th International Conference on Autonomic and Trusted Computing (ATC-08)*, number 5060 in LNCS, pages 219–233. Springer, June 2010. doi: 10.1007/978-3-540-69295-9_19.

PTV AG. *VISSIM 5.10 User Manual*, 2008.

N. Pyrgiotis, K.M. Malone, and A. Odoni. Modelling Delay Propagation within an Airport Network. *Transportation Research Part C: Emerging Technologies*, (0):–, 2011. ISSN 0968-090X. doi: 10.1016/j.trc.2011.05.017.

B. Raney and K. Nagel. An improved framework for large-scale multi-agent simulations of travel behaviour. In P. Rietveld, B. Jourquin, and K. Westin, editors, *Towards better performing European Transportation Systems*, pages 305–347. Routledge, London, 2006.

B. K. Raney. *Learning Framework for Large-Scale Multi-Agent Simulations*. PhD thesis, Swiss Federal Institute of Technology (ETH) Zurich, 2005.

M. Rieser. matsimJ An Overview of the new MATSim Implementation in Java. Slides, presented at MATSim Seminar 2006, Villa Garbald, 10 2006.

M. Rieser. *Adding transit to an agent-based transportation simulation concepts and implementation*. PhD thesis, TU Berlin, 2010. Also VSP WP 10-05, see `www.vsp.tu-berlin.de/publications`.

M. Rieser, U. Beuck, M. Balmer, and K. Nagel. Modelling and simulation of a morning reaction to an evening toll. In *Innovations in Travel Modeling (ITM) '08*, Portland, Oregon, June 2008. Also VSP WP 08-01, see `www.vsp.tu-berlin.de/publications`.

M. Rieser, D. Grether, and K. Nagel. Adding mode choice to a multi-agent transport simulation. *Transportation Research Record*, 2132:50–58, 2009. doi: 10.3141/2132-06.

D. I. Robertson. TRANSYT Method for Area Traffic Control. *Traffic Engineering and Control*, 11(6):276–281, 1969. URL `http://trid.trb.org/view.aspx?id=116630`.

D.I. Robertson and R.D. Bretherton. Optimizing Networks of Traffic Signals in Real Time – the SCOOT Method. *Vehicular Technology, IEEE Transactions on*, 40(1):11 –15, Feb 1991. ISSN 0018-9545. doi: 10.1109/25.69966.

D. Röder. Modellierung und Simulation einer adaptiven Steuerung für Lichtsignalanlagen in Queue-Simulationen. Bachelor's thesis, TU Berlin, Institute for Land and Sea Transport Systems, Berlin, Germany, 2010.

T. Roughgarden. On the severity of braess's paradox: Designing networks for selfish users is hard. *Journal of Computer and System Sciences*, 72(5):922–953, 2006. ISSN 0022-0000. doi: 10.1016/j.jcss.2005.05.009. Special Issue on {FOCS} 2001.

S. Russel and P. Norvig. *Artificial Intelligence – A Modern Approach*. Pearson Education, Upper Saddle River, New Jersey 07458, 3 edition, 2010.

R. Sanders and D. Kelly. Dealing with risk in scientific software development. *Software, IEEE*, 25(4):21–28, 2008. ISSN 0740-7459. doi: 10.1109/MS.2008.84.

Schlothauer & Wauer Ingenieurgesellschaft für Straßenverkehr mbH & Co. KG. Das verkehrsabhängige LSA-Steuerverfahren SYLVIA+. *Straßenverkehrstechnik*, 12:790–791, 2010.

Schlothauer & Wauer Ingenieurgesellschaft für Straßenverkehr mbH & Co. KG. SYLVIA+ short description. www.schlothauer.de/en/Control_Systems.html, 2011. URL `http://www.schlothauer.de/docs/SYLVIA_Short-Description.pdf`. Website version: 17-01-2011.

D. Schultes. *Route Planning in Road Networks*. PhD thesis, TH Karlsruhe, 2008.

P.M. Simon, J. Esser, and K. Nagel. Simple queueing model applied to the city of Portland. *International Journal of Modern Physics*, 10(5):941–960, 1999. doi: 10.1142/S0129183199000747.

M.J. Smith. The existence, uniqueness and stability of traffic equilibria. *Transportation Research Part B: Methodological*, 13(4):295–304, 1979a. ISSN 0191-2615. doi: 10.1016/0191-2615(79)90022-5.

M.J. Smith. Traffic control and route-choice; a simple example. *Transportation Research Part B: Methodological*, 13(4):289–294, 1979b. ISSN 0191-2615. doi: 10.1016/0191-2615(79)90021-3.

M.J. Smith. A new dynamic traffic model and the existence and calculation of dynamic user equilibria on congested capacity-constrained road networks. *Transportation Research Part B: Methodological*, 27(1):49–63, 1993. ISSN 0191-2615. doi: 10.1016/0191-2615(93)90011-X.

M.J. Smith and T. van Vuren. Traffic equilibrium with responsive traffic control. *Transportation Science*, 27(2):118–132, 1993. doi: 10.1287/trsc.27.2.118.

W.P. Stevens, G.J. Myers, and L.L. Constantive. Structured design. *IBM Systems Journal*, 13(2):115–139, 1974. ISSN 0018-8670.

M. Strehler. *Signalized flows – optimizing traffic signals and guideposts and related network flow problems*. PhD thesis, BTU Cottbus, 2012.

D. Sweet, V. Manikonda, J. Aronson, K. Roth, and M. Blake. Fast-Time Simulation System for Analysis of Advanced Air Transportation Concepts. In *AIAA Modeling and Simulation Technologies Conference and Exhibit*, number AIAA-2002-4593, Monterey, CA, 08 2002.

H. Taale. *Integrated Anticipatory Control of Road Networks – A game-theoretical approach*. PhD thesis, Technische Universiteit Delft, 2008.

H. Taale and H.J. Van Zuylen. The effects of anticipatory traffic control for several small networks. In *Transportation Research Board 82nd Annual Meeting*, Washington D.C., 2003.

T. Titze. Umsetzung des Steuerverfahrens SYLVIA mit dem Programmpaket LISA+. Diplomarbeit (Diploma Thesis), TU Berlin, Institute for Land and Sea Transport Systems, Berlin, Germany, 2008.

K. Train. *Discrete choice methods with simulation*. Cambridge University Press, 2003.

C.M. Treczka. Modellierung und Simulation der Luftverkehrsnachfrage im deutschen Luftraum. Bachelor's thesis, TU Berlin, Institute for Land and Sea Transport Systems, Berlin, Germany, 01 2012.

H.J. van Zuylen and H. Taale. Urban networks with ring roads: Two-level, three-player game. *Transportion Research Record*, 1894:180–187, 2004.

P. Wagner, E. Brockfeld, N.H. Gartner, and A. Sohr. Fundamental diagram of traffic flows on urban roads. *Transportation Research Record: Journal of the Transportation Research Board*, 2124(1):213–221, 12 2009. doi: 10.3141/2124-21.

R. Waraich, D. Charypar, M. Balmer, and K. Axhausen. Performance improvements for large scale traffic simulation in MATSim. Technical report, ETH Zürich, IVT, 2009a.

R.A. Waraich, M.D. Galus, C. Dobler, M. Balmer, G. Andersson, and K.W. Axhausen. Plug-in hybrid electric vehicles and smart grid. Investigations based on a microsimulation. In *Proceedings of The 12th Conference of the International Association for Travel Behaviour Research (IATBR)*, Jaipur, India, 2009b. URL `www.iatbr.org`.

F.V. Webster. Traffic Signal Settings. In *Road Research Technical Paper No. 39*, pages 1–45. Her Majesty's Stationery Office, 1961.

G. Weiss, editor. *Multiagent Systems. A modern approach to distributed artificial intelligence*. The MIT Press, 1999.

R. Wiedemann. Simulation des Strassenverkehrsflusses. Schriftenreihe Heft 8, Institute for Transportation Science, University of Karlsruhe, Germany, 1974.

D. Wiethölter, D. Bogai, and J. Carstensen. Pendlerbericht Berlin-Brandenburg 2009. Technical report, Institut für Arbeitsmarkt- und Berufsforschung, 2010. URL `http://www.iab.de/238/section.aspx/Publikation/k100921n04`.

N. Zheng, R.A. Waraich, K.W. Axhausen, and N. Geroliminis. A dynamic cordon pricing scheme combining the macroscopic fundamental diagram and an agent-based traffic model. *Transportation Research Part A: Policy and Practice*, 46(8):1291–1303, 2012. ISSN 0965-8564. doi: 10.1016/j.tra.2012.05.006.

Acknowledgements

I am indebted to my family, Katharina, and my friends, who supported me patiently throughout the years with advice, encouragement, and diversion. I would like to express my gratitude to my supervisor Prof. Dr. Kai Nagel, who gave me the opportunity to work on the topics covered in this thesis. His guidance and advice proved consistently helpful and enlightening, and provided me with a solid overview of transportation science. I would also like to thank my supervisors, Prof. Dr. Kai Nagel and Dr. Peter Wagner, for their interest and helpful tips to complete this work.

My gratitude to Prof. Dr. Kai Nagel and Dr. Marcel Rieser who spent much effort to "clean up" the MATSim code. Further, I am indebted to my colleages at VSP, TU Berlin and the development team of MATSim. Without this excellent team, this work would not have been possible.

I would also like to thank the many students for developing prototypes and preliminary results (Titze, 2008; Neumann, 2008; Röder, 2010; Bischoff, 2010; Graf, 2013; Fürbas, 2011; Böhme, 2011; Treczka, 2012; Krüger, 2012). These prototypes were then improved and developed further; the thesis describes the overall result.

I would like to thank Prof. Dr. Ekkehard Köhler and Dr. Martin Strehler at BTU Cottbus, for the optimization model and traffic signal data, as well as for the very pleasant and fruitful cooperation.

My further acknowledgements to all providers of data used in this thesis. Without accessible data, suited for machine processing, this work would not have been possible. I would also like to thank the computeserver team of TU Berlin's Department of Mathematics for maintaining the cluster machines that were used to compute the results of this work.

I would like to publically recognize all the other open source software developers out there, whom I hold in high esteem. The variety of great open source tools applied to gather the results of this thesis is wide, and I would like to thank all anonymous authors.

This work was funded in part by the German Ministry for Education and Research (BMBF), under Grants Number 03NAPI4 ("Advest") and the German research society,

DFG, under the grant “Optimization and network wide analysis of traffic signal control”.

List of Figures

List of Tables

Appendix A

Simulation Setups and Configuration

In the following, the technical documentation for the results presented within this thesis is provided. This includes a revision that points to the subversion repository, currently hosted at sourceforge (`svn://svn.code.sf.net/p/matsim/source/matsim-source`). In conjunction with maven as build management, on the command line `mvn assembly:assembly` provides a build of the software. Further, the path of the executable Java class is provided. The `.log` file then provides an overview of defaults and parameters of extensions that are not used at all. Names apply to the given subversion revision and might have changed in the up-to-date head revision. The (sub-)version management system allows to trace these redefinitions. For each study, an example configuration is listed in conjunction with the parameter that is varied. With a high probability, the configuration is not accepted by the current version of MATSim, but by the provided code revision. Further explanations for variation of simulation setup can be found in the corresponding chapters of the thesis.

A.1. Network Effects of Traffic Signal Control

A.1.1. Illustrative Example

MATSim revision: r20219 (2012-07-17 18:29:03)

Base Case & Direct Route Expansion

Executable: `playground.dgrether.daganzo2012.Daganzo2012Run`

The network is varied for the two simulation runs.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE config SYSTEM "http://www.matsim.org/files/dtd/config_v1.dtd"
  [
   <!ENTITY INPUTBASE "/net/homes2/extern/grether/netils3/shared-svn/studies/
       dgrether/daganzo2012/scenario_2/">
   <!ENTITY OUTPUTBASE  "/net/homes2/extern/grether/netils3/matsimOutput/
       run1574/">
    ]
>
<config>
  <module name="controler" >
    <param name="enableLinkToLinkRouting" value="false" />
    <param name="eventsFileFormat" value="xml" />
    <param name="firstIteration" value="0" />
    <param name="lastIteration" value="2000" />
    <param name="mobsim" value="qsim" />
    <param name="outputDirectory" value="&OUTPUTBASE;" />
    <param name="runId" value="1574" />
    <param name="writeEventsInterval" value="1000" />
    <param name="writePlansInterval" value="1000" />
    <param name="writeSnapshotsInterval" value="1000" />
  </module>

  <module name="global" >
    <param name="coordinateSystem" value="Atlantis" />
    <param name="numberOfThreads" value="1" />
    <param name="randomSeed" value="4711" />
  </module>

  <module name="network" >
    <param name="inputNetworkFile" value="&INPUTBASE;network21.xml" />
  </module>

  <module name="otfvis" >
    <param name="linkwidthIsProportionalTo" value="numberOfLanes" />
  </module>

  <module name="planCalcScore" >
    <param name="BrainExpBeta" value="2.0" />
    <param name="PathSizeLogitBeta" value="1.0" />
    <param name="activityClosingTime_0" value="undefined" />
    <param name="activityEarliestEndTime_0" value="undefined" />
    <param name="activityLatestStartTime_0" value="undefined" />
    <param name="activityMinimalDuration_0" value="undefined" />
    <param name="activityOpeningTime_0" value="undefined" />
    <param name="activityPriority_0" value="1.0" />
    <param name="activityType_0" value="h" />
    <param name="activityTypicalDuration_0" value="24:00:00" />
    <param name="constantBike" value="0.0" />
```

```
  <param name="constantCar" value="0.0" />
  <param name="constantPt" value="0.0" />
  <param name="constantWalk" value="0.0" />
  <param name="earlyDeparture" value="-0.0" />
  <param name="lateArrival" value="0.0" />
  <param name="learningRate" value="1.0" />
  <param name="marginalUtilityOfMoney" value="1.0" />
  <param name="marginalUtlOfDistanceWalk" value="0.0" />
  <param name="monetaryDistanceCostRateCar" value="0.0" />
  <param name="monetaryDistanceCostRatePt" value="0.0" />
  <param name="performing" value="0.0" />
  <param name="traveling" value="-6.0" />
  <param name="travelingBike" value="-6.0" />
  <param name="travelingPt" value="-6.0" />
  <param name="travelingWalk" value="-6.0" />
  <param name="utilityOfLineSwitch" value="-1.0" />
  <param name="waiting" value="0.0" />
</module>

<module name="plans" >
  <param name="inputPlansFile" value="&INPUTBASE;/plans.xml" />
</module>

<module name="qsim" >
  <param name="flowCapacityFactor" value="1.0" />
  <param name="numberOfThreads" value="1" />
  <param name="removeStuckVehicles" value="false" />
  <param name="stuckTime" value="100.0" />
  <param name="simStarttimeInterpretation" value="
      maxOfStarttimeAndEarliestActivityEnd" />
  <param name="snapshotStyle" value="queue" />
  <param name="snapshotperiod" value="00:00:10" />
  <param name="startTime" value="00:00:00" />
  <param name="storageCapacityFactor" value="1.0" />
  <param name="vehicleBehavior" value="exception" />
</module>

<module name="strategy" >
  <param name="ModuleProbability_1" value="0.9" />
  <param name="Module_1" value="ChangeExpBeta" />
  <param name="ModuleProbability_2" value="0.1" />
  <param name="Module_2" value="ReRoute" />
  <param name="maxAgentPlanMemorySize" value="5" />
</module>

<module name="travelTimeCalculator" >
  <param name="calculateLinkToLinkTravelTimes" value="false" />
  <param name="calculateLinkTravelTimes" value="true" />
  <param name="travelTimeBinSize" value="1" />
```

```
    <param name="travelTimeCalculator" value="TravelTimeCalculatorArray" />
  </module>

  <module name="vspExperimental" >
    <param name="activityDurationInterpretation" value="minOfDurationAndEndTime"
        />
    <param name="vspDefaultsCheckingLevel" value="abort" />
    <param name="usingOpportunityCostOfTimeForPtRouting" value="true" />
  </module>
</config>
```

Adding an Adaptive Signal

Executable: `playground.dgrether.daganzo2012.Daganzo2012SimpleAdaptiveRun`

The parameter `"initialRedOn4"` is varied.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE config SYSTEM "http://www.matsim.org/files/dtd/config_v1.dtd"
  [
   <!ENTITY INPUTBASE "/net/homes2/extern/grether/netils3/shared-svn/studies/
       dgrether/daganzo2012/scenario_2/">
   <!ENTITY OUTPUTBASE  "/net/homes2/extern/grether/netils3/matsimOutput/
       run1579/">
    ]
>
<config>
  <module name="controler" >
    <param name="enableLinkToLinkRouting" value="false" />
    <param name="eventsFileFormat" value="xml" />
    <param name="firstIteration" value="0" />
    <param name="lastIteration" value="2000" />
    <param name="mobsim" value="qsim" />
    <param name="outputDirectory" value="&OUTPUTBASE;" />
    <param name="runId" value="1579" />
    <param name="writeEventsInterval" value="1000" />
    <param name="writePlansInterval" value="1000" />
    <param name="writeSnapshotsInterval" value="1000" />
  </module>

  <module name="daganzo2012">
    <param name="initialRedOn4" value="0" />
  </module>

  <module name="global" >
    <param name="coordinateSystem" value="Atlantis" />
    <param name="numberOfThreads" value="1" />
    <param name="randomSeed" value="4711" />
```

```
</module>

<module name="network" >
  <param name="inputNetworkFile" value="&INPUTBASE;network22.xml" />
</module>

<module name="otfvis" >
  <param name="linkwidthIsProportionalTo" value="numberOfLanes" />
</module>

<module name="planCalcScore" >
  <param name="BrainExpBeta" value="2.0" />
  <param name="PathSizeLogitBeta" value="1.0" />
  <param name="activityClosingTime_0" value="undefined" />
  <param name="activityEarliestEndTime_0" value="undefined" />
  <param name="activityLatestStartTime_0" value="undefined" />
  <param name="activityMinimalDuration_0" value="undefined" />
  <param name="activityOpeningTime_0" value="undefined" />
  <param name="activityPriority_0" value="1.0" />
  <param name="activityType_0" value="h" />
  <param name="activityTypicalDuration_0" value="24:00:00" />
  <param name="constantBike" value="0.0" />
  <param name="constantCar" value="0.0" />
  <param name="constantPt" value="0.0" />
  <param name="constantWalk" value="0.0" />
  <param name="earlyDeparture" value="-0.0" />
  <param name="lateArrival" value="0.0" />
  <param name="learningRate" value="1.0" />
  <param name="marginalUtilityOfMoney" value="1.0" />
  <param name="marginalUtlOfDistanceWalk" value="0.0" />
  <param name="monetaryDistanceCostRateCar" value="0.0" />
  <param name="monetaryDistanceCostRatePt" value="0.0" />
  <param name="performing" value="0.0" />
  <param name="traveling" value="-6.0" />
  <param name="travelingBike" value="-6.0" />
  <param name="travelingPt" value="-6.0" />
  <param name="travelingWalk" value="-6.0" />
  <param name="utilityOfLineSwitch" value="-1.0" />
  <param name="waiting" value="0.0" />
</module>

<module name="plans" >
  <param name="inputPlansFile" value="&INPUTBASE;/plans.xml" />
</module>

<module name="qsim" >
  <param name="flowCapacityFactor" value="1.0" />
  <param name="numberOfThreads" value="1" />
  <param name="removeStuckVehicles" value="false" />
```

```
    <param name="stuckTime" value="100.0" />
    <param name="simStarttimeInterpretation" value="
        maxOfStarttimeAndEarliestActivityEnd" />
    <param name="snapshotStyle" value="queue" />
    <param name="snapshotperiod" value="00:00:10" />
    <param name="startTime" value="00:00:00" />
    <param name="storageCapacityFactor" value="1.0" />
    <param name="vehicleBehavior" value="exception" />
  </module>

  <module name="strategy" >
    <param name="ModuleProbability_1" value="0.9" />
    <param name="Module_1" value="ChangeExpBeta" />
    <param name="ModuleProbability_2" value="0.1" />
    <param name="Module_2" value="ReRoute" />
    <param name="maxAgentPlanMemorySize" value="5" />
  </module>

  <module name="travelTimeCalculator" >
    <param name="calculateLinkToLinkTravelTimes" value="false" />
    <param name="calculateLinkTravelTimes" value="true" />
    <param name="travelTimeBinSize" value="1" />
    <param name="travelTimeCalculator" value="TravelTimeCalculatorArray" />
  </module>

  <module name="vspExperimental" >
    <param name="activityDurationInterpretation" value="minOfDurationAndEndTime"
        />
    <param name="vspDefaultsCheckingLevel" value="abort" />
    <param name="usingOpportunityCostOfTimeForPtRouting" value="true" />
  </module>
</config>
```

A.1.2. Cottbus Scenario

MATSim revision: r20407 (2012-07-27 15:05:10)

Executable: `playground.dgrether.DgController`

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE config SYSTEM "http://www.matsim.org/files/dtd/config_v1.dtd"
  [
   <!ENTITY INPUTBASE "/net/homes2/extern/grether/netils3/shared-svn/studies/
       dgrether/cottbus/cottbus_feb_fix/">
   <!ENTITY OUTPUTBASE  "/net/homes2/extern/grether/netils3/matsimOutput/
       run1712/">
  ]
>
```

```
<config>
  <module name="global" >
    <param name="numberOfThreads" value="2" />
  </module>

  <module name="controler" >
    <param name="enableLinkToLinkRouting" value="true" />
    <param name="eventsFileFormat" value="xml" />
    <param name="firstIteration" value="0" />
    <param name="lastIteration" value="1000" />
    <param name="outputDirectory" value="&OUTPUTBASE;" />
    <param name="runId" value="1712" />
    <param name="writeEventsInterval" value="100" />
    <param name="writePlansInterval" value="100" />
    <param name="writeSnapshotsInterval" value="100" />
    <param name="mobsim" value="qsim" />
    <param name="snapshotFormat" value="otfvis" />
  </module>

  <module name="network" >
    <param name="inputNetworkFile" value="&INPUTBASE;network_wgs84_utm33n.xml.gz
        " />
    <param name="laneDefinitionsFile" value="&INPUTBASE;lanes.xml" />
  </module>

  <module name="planCalcScore" >
    <param name="learningRate" value="1.0" />
    <param name="BrainExpBeta" value="2.0" />

    <param name="lateArrival" value="-18" />
    <param name="performing" value="+6" />
    <param name="traveling" value="-6" />

    <param name="activityType_0" value="home" /> <!-- home -->
    <param name="activityPriority_0" value="1" />
    <param name="activityTypicalDuration_0" value="15:30:00" />

    <param name="activityType_1" value="work" /> <!-- work -->
    <param name="activityPriority_1" value="1" />
    <param name="activityTypicalDuration_1" value="08:30:00" />
    <param name="activityLatestStartTime_1" value="09:00:00" />
    <param name="activityOpeningTime_1" value="07:00:00" />
    <param name="activityClosingTime_1" value="18:00:00" />

    <param name="activityType_2" value="fb" /> <!-- football -->
    <param name="activityPriority_2" value="1" />
    <param name="activityTypicalDuration_2" value="02:30:00" />
    <param name="activityLatestStartTime_2" value="17:30:00" />
    <param name="activityOpeningTime_2" value="17:00:00" />
```

```
  <param name="activityClosingTime_2" value="20:15:00" />
</module>

<module name="plans" >
  <param name="inputPlansFile" value="&INPUTBASE;
      cb_spn_gemeinde_nachfrage_landuse/
      commuter_population_wgs84_utm33n_car_only.xml.gz" />
</module>

<module name="qsim" >
  <param name="numberOfThreads" value="1" />
  <param name="removeStuckVehicles" value="false" />
  <param name="snapshotStyle" value="queue" />
  <param name="snapshotperiod" value="00:05:00" />
  <param name="stuckTime" value="100.0" />
  <param name="flowCapacityFactor" value="0.70" />
  <param name="storageCapacityFactor" value="0.70" />
</module>

<module name="scenario" >
  <param name="useLanes" value="true" />
  <param name="useSignalsystems" value="true" />
</module>

<module name="signalsystems" >
  <param name="ambertimes" value="&INPUTBASE;amber_times.xml.gz" />
  <param name="signalcontrol" value="&INPUTBASE;signal_control.xml" />
  <param name="signalgroups" value="&INPUTBASE;signal_groups.xml" />
  <param name="signalsystems" value="&INPUTBASE;signal_systems.xml" />
</module>

<module name="strategy" >
  <param name="maxAgentPlanMemorySize" value="4" />
  <param name="Module_1" value="ChangeExpBeta" />
  <param name="ModuleProbability_1" value="0.8" />
  <param name="Module_2" value="ReRoute" />
  <param name="ModuleProbability_2" value="0.1" />
  <param name="Module_3" value="TimeAllocationMutator" />
  <param name="ModuleProbability_3" value="0.1" />
  <param name="ModuleDisableAfterIteration_2" value="500" />
  <param name="ModuleDisableAfterIteration_3" value="500" />
</module>

<module name="travelTimeCalculator" >
  <param name="calculateLinkToLinkTravelTimes" value="true" />
  <param name="calculateLinkTravelTimes" value="true" />
</module>

<module name="TimeAllocationMutator" >
```

```
    <param name="mutationRange" value="7200" />
  </module>

  <module name="vspExperimental" >
    <param name="activityDurationInterpretation" value="endTimeOnly" />
    <param name="vspDefaultsCheckingLevel" value="abort" />
    <param name="usingOpportunityCostOfTimeForPtRouting" value="true" />
  </module>
</config>
```

A.1.3. Optimization and Network Wide Analysis of Traffic Signal Control

Density, Speed, Flow

MATSim revision: r25540 (2013-08-12 17:00:00)

Executable: `playground.dgrether.DgController`

The parameters `"flowCapacityFactor"` and `"storageCapacityFactor"` are varied.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE config SYSTEM "http://www.matsim.org/files/dtd/config_v1.dtd"
[
 <!ENTITY INPUTBASE "/net/homes2/extern/grether/netils3/shared-svn/studies/
     dgrether/cottbus/cottbus_feb_fix/">
 <!ENTITY PROJECTSBASE "/net/homes2/extern/grether/netils3/shared-svn/projects/
     cottbus/cb2ks2010/">
 <!ENTITY OUTPUTBASE  "/net/homes2/extern/grether/netils3/matsimOutput/run1950/
     ">
]
>
<config>
  <module name="global" >
    <param name="numberOfThreads" value="2" />
  </module>

  <module name="controler" >
    <param name="enableLinkToLinkRouting" value="true" />
    <param name="eventsFileFormat" value="xml" />
    <param name="firstIteration" value="0" />
    <param name="lastIteration" value="1000" />
    <param name="outputDirectory" value="&OUTPUTBASE;" />
    <param name="runId" value="1950" />
    <param name="writeEventsInterval" value="100" />
    <param name="writePlansInterval" value="100" />
    <param name="writeSnapshotsInterval" value="100" />
    <param name="mobsim" value="qsim" />
    <param name="snapshotFormat" value="otfvis" />
```

```
</module>

<module name="qsim" >
  <param name="numberOfThreads" value="1" />
  <param name="removeStuckVehicles" value="false" />
  <param name="snapshotStyle" value="queue" />
  <param name="snapshotperiod" value="00:05:00" />
  <param name="stuckTime" value="100.0" />
  <param name="flowCapacityFactor" value="0.50" />
  <param name="storageCapacityFactor" value="0.50" />
  <param name="nodeOffset" value="40.0" />
</module>

<module name="otfvis" >
  <param name="agentSize" value="70.0" />
  <param name="linkWidth" value="30.0" />
  <param name="linkwidthIsProportionalTo" value="numberOfLanes" />
  <param name="showTeleportation" value="false" />
</module>

<module name="scenario" >
  <param name="useLanes" value="true" />
  <param name="useSignalsystems" value="true" />
</module>


<module name="network" >
  <param name="inputNetworkFile" value="&INPUTBASE;network_wgs84_utm33n.xml.gz
      " />
  <param name="laneDefinitionsFile" value="&INPUTBASE;lanes.xml" />
</module>

<module name="plans" >
  <param name="inputPlansFile" value="&INPUTBASE;
      cb_spn_gemeinde_nachfrage_landuse/
      commuter_population_wgs84_utm33n_car_only.xml.gz" />
</module>

<module name="signalsystems" >
  <param name="ambertimes" value="&INPUTBASE;amber_times.xml.gz" />
  <param name="signalcontrol" value="&INPUTBASE;signal_control_no_13.xml" />
  <param name="signalgroups" value="&INPUTBASE;signal_groups_no_13.xml" />
  <param name="signalsystems" value="&INPUTBASE;signal_systems_no_13.xml" />
</module>

<module name="planCalcScore" >
  <param name="learningRate" value="1.0" />
  <param name="BrainExpBeta" value="2.0" />
```

```
  <param name="lateArrival" value="-18" />
  <param name="performing" value="+6" />
  <param name="traveling" value="-6" />

  <param name="activityType_0" value="home" />
  <param name="activityPriority_0" value="1" />
  <param name="activityTypicalDuration_0" value="15:30:00" />

  <param name="activityType_1" value="work" />
  <param name="activityPriority_1" value="1" />
  <param name="activityTypicalDuration_1" value="08:30:00" />
  <param name="activityLatestStartTime_1" value="09:00:00" />
  <param name="activityOpeningTime_1" value="07:00:00" />
  <param name="activityClosingTime_1" value="18:00:00" />

  <param name="activityType_2" value="fb" />
  <param name="activityPriority_2" value="1" />
  <param name="activityTypicalDuration_2" value="02:30:00" />
  <param name="activityLatestStartTime_2" value="17:30:00" />
  <param name="activityOpeningTime_2" value="17:00:00" />
  <param name="activityClosingTime_2" value="20:15:00" />
</module>

<module name="strategy" >
  <param name="maxAgentPlanMemorySize" value="4" />
  <param name="Module_1" value="ChangeExpBeta" />
  <param name="ModuleProbability_1" value="0.8" />
  <param name="Module_2" value="ReRoute" />
  <param name="ModuleProbability_2" value="0.1" />
  <param name="Module_3" value="TimeAllocationMutator" />
  <param name="ModuleProbability_3" value="0.1" />
  <param name="ModuleDisableAfterIteration_2" value="500" />
  <param name="ModuleDisableAfterIteration_3" value="500" />
</module>

<module name="travelTimeCalculator" >
  <param name="calculateLinkToLinkTravelTimes" value="true" />
  <param name="calculateLinkTravelTimes" value="true" />
</module>

<module name="TimeAllocationMutator" >
  <param name="mutationRange" value="7200" />
</module>

<module name="vspExperimental" >
  <param name="activityDurationInterpretation" value="tryEndTimeThenDuration"
      />
  <param name="vspDefaultsCheckingLevel" value="abort" />
  <param name="removingUnnecessaryPlanAttributes" value="true" />
```

```
  </module>
</config>
```

A.1.4. Results

MATSim revision: r20407 (2012-07-27 15:05:10)

Executable: `playground.dgrether.DgController`

The parameter `"signalcontrol"` is varied.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE config SYSTEM "http://www.matsim.org/files/dtd/config_v1.dtd"
  [
   <!ENTITY INPUTBASE "/net/homes2/extern/grether/netils3/shared-svn/studies/
       dgrether/cottbus/cottbus_feb_fix/">
   <!ENTITY OUTPUTBASE  "/net/homes2/extern/grether/netils3/matsimOutput/
       run1912/">
   <!ENTITY PROJECTSBASE "/net/homes2/extern/grether/netils3/shared-svn/
       projects/cottbus/cb2ks2010/">
   <!ENTITY RUNBASE   "/net/homes2/extern/grether/netils3/runs-svn/">
  ]
>
<config>
  <module name="scenario" >
    <param name="useLanes" value="true" />
    <param name="useSignalsystems" value="true" />
  </module>

  <module name="network" >
    <param name="inputNetworkFile" value="&INPUTBASE;network_wgs84_utm33n.xml.gz
        " />
    <param name="laneDefinitionsFile" value="&INPUTBASE;lanes.xml" />
  </module>

  <module name="plans" >
    <param name="inputPlansFile" value="&RUNBASE;run1712/1712.output_plans.xml.
        gz" />
  </module>

  <module name="signalsystems" >
    <param name="ambertimes" value="&INPUTBASE;amber_times.xml.gz" />
    <param name="signalcontrol" value="&PROJECTSBASE;2013-07-31_minflow_50/
        merged_signal_control_ksm_50m_sol_ksm_50a_sol.xml" />
    <param name="signalgroups" value="&INPUTBASE;signal_groups_no_13.xml" />
    <param name="signalsystems" value="&INPUTBASE;signal_systems_no_13.xml" />
  </module>
```

```
<module name="global" >
  <param name="numberOfThreads" value="2" />
</module>

<module name="controler" >
  <param name="enableLinkToLinkRouting" value="true" />
  <param name="eventsFileFormat" value="xml" />
  <param name="firstIteration" value="1000" />
  <param name="lastIteration" value="2000" />
  <param name="outputDirectory" value="&OUTPUTBASE;" />
  <param name="runId" value="1912" />
  <param name="writeEventsInterval" value="100" />
  <param name="writePlansInterval" value="100" />
  <param name="writeSnapshotsInterval" value="100" />
  <param name="mobsim" value="qsim" />
  <param name="snapshotFormat" value="otfvis" />
</module>

<module name="strategy" >
  <param name="maxAgentPlanMemorySize" value="4" />
  <param name="Module_1" value="ChangeExpBeta" />
  <param name="ModuleProbability_1" value="0.9" />
  <param name="Module_2" value="ReRoute" />
  <param name="ModuleProbability_2" value="0.1" />
  <param name="ModuleDisableAfterIteration_2" value="1500" />
  <!--<param name="Module_3" value="TimeAllocationMutator" />
  <param name="ModuleProbability_3" value="0.1" />
  <param name="ModuleDisableAfterIteration_3" value="1500" />-->
</module>

<module name="qsim" >
  <param name="numberOfThreads" value="1" />
  <param name="removeStuckVehicles" value="false" />
  <param name="snapshotStyle" value="queue" />
  <param name="snapshotperiod" value="00:05:00" />
  <param name="stuckTime" value="100.0" />
  <param name="flowCapacityFactor" value="0.70" />
  <param name="storageCapacityFactor" value="0.70" />
</module>


<module name="planCalcScore" >
  <param name="learningRate" value="1.0" />
  <param name="BrainExpBeta" value="2.0" />

  <param name="lateArrival" value="-18" />
  <param name="performing" value="+6" />
  <param name="traveling" value="-6" />
```

```
    <param name="activityType_0" value="home" />
    <param name="activityPriority_0" value="1" />
    <param name="activityTypicalDuration_0" value="15:30:00" />

    <param name="activityType_1" value="work" />
    <param name="activityPriority_1" value="1" />
    <param name="activityTypicalDuration_1" value="08:30:00" />
    <param name="activityLatestStartTime_1" value="09:00:00" />
    <param name="activityOpeningTime_1" value="07:00:00" />
    <param name="activityClosingTime_1" value="18:00:00" />

    <param name="activityType_2" value="fb" />
    <param name="activityPriority_2" value="1" />
    <param name="activityTypicalDuration_2" value="02:30:00" />
    <param name="activityLatestStartTime_2" value="17:30:00" />
    <param name="activityOpeningTime_2" value="17:00:00" />
    <param name="activityClosingTime_2" value="20:15:00" />
  </module>

  <module name="travelTimeCalculator" >
    <param name="calculateLinkToLinkTravelTimes" value="true" />
    <param name="calculateLinkTravelTimes" value="true" />
  </module>


  <module name="TimeAllocationMutator" >
    <param name="mutationRange" value="7200" />
  </module>

  <module name="vspExperimental" >
    <param name="activityDurationInterpretation" value="endTimeOnly" />
    <param name="vspDefaultsCheckingLevel" value="abort" />
    <param name="usingOpportunityCostOfTimeForPtRouting" value="true" />
  </module>
</config>
```

A.2. Modeling and Simulation of Air Transport Systems

A.2.1. Air Transport Technology

MATSim revision: r20407 (2012-08-03 15:05:10)

Executable: `playground.fuerbas.SfAirController`, for delay via random draw `playground.fuerbas.DgFlightDelayController`

The parameter `"inputNetworkFile"` is varied.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE config SYSTEM "http://www.matsim.org/files/dtd/config_v1.dtd"
  [
    <!ENTITY INPUTBASE "/net/ils3/dgrether/shared-svn/studies/countries/eu/
        flight/dg_oag_tuesday_flight_model_2_runways_3600vph/">
    <!ENTITY OUTPUTBASE  "/net/ils3/dgrether/matsimOutput/run1812/">
  ]
>
<config>
  <module name="global" >
    <param name="coordinateSystem" value="EPSG:3395" />
    <param name="numberOfThreads" value="1" />
    <param name="randomSeed" value="4711" />
  </module>

  <module name="network" >
    <param name="inputNetworkFile" value="&INPUTBASE;air_network.xml" />
  </module>


  <module name="plans" >
    <param name="inputPlansFile" value="null" />
  </module>


  <module name="scenario" >
    <param name="useTransit" value="true" />
    <param name="useVehicles" value="true" />
  </module>

  <module name="transit" >
    <param name="transitModes" value="pt" />
    <param name="transitScheduleFile" value="&INPUTBASE;flight_transit_schedule.
        xml" />
    <param name="vehiclesFile" value="&INPUTBASE;flight_transit_vehicles.xml" />
  </module>

  <module name="controler" >
    <param name="eventsFileFormat" value="xml" />
    <param name="firstIteration" value="0" />
    <param name="lastIteration" value="0" />
    <param name="outputDirectory" value="&OUTPUTBASE;" />
    <param name="runId" value="1812" />
    <param name="writeEventsInterval" value="1" />
    <param name="snapshotFormat" value="otfvis" />
    <param name="mobsim" value="qsim" />
  </module>

  <module name="qsim" >
```

```
    <param name="flowCapacityFactor" value="1.0" />
    <param name="storageCapacityFactor" value="1.0" />
    <param name="removeStuckVehicles" value="false" />
    <param name="simStarttimeInterpretation" value="
        maxOfStarttimeAndEarliestActivityEnd" />
    <param name="snapshotStyle" value="queue" />
    <param name="snapshotperiod" value="00:05:00" />
    <param name="stuckTime" value="100.0" />
  </module>


  <module name="otfvis" >
    <param name="linkWidth" value="50.0" />
    <param name="showTeleportation" value="false" />
  </module>

  <module name="planCalcScore" >
    <param name="BrainExpBeta" value="2.0" />
    <param name="performing" value="6.0" />
    <param name="traveling" value="-6.0" />
    <param name="travelingPt" value="-6.0" />
    <param name="lateArrival" value="-18.0" />
    <param name="travelingWalk" value="-6.0" />
  </module>

  <module name="strategy" >
    <param name="ModuleProbability_1" value="0.9" />
    <param name="ModuleProbability_2" value="0.1" />
    <param name="Module_1" value="ChangeExpBeta" />
    <param name="Module_2" value="ReRoute" />
    <param name="maxAgentPlanMemorySize" value="4" />
  </module>

  <module name="TimeAllocationMutator">
    <param name="mutationRange" value="7200.0" />
  </module>

  <module name="vspExperimental" >
    <param name="usingOpportunityCostOfTimeForPtRouting" value="true" />
    <param name="vspDefaultsCheckingLevel" value="abort" />
    <param name="removingUnnecessaryPlanAttributes" value="true" />-->
  </module>
</config>
```

A.2.2. Passenger Demand

MATSim revision : r24284 (2013-05-24 18:09:32)

Executable: `playground.fuerbas.DgFlightControllerStucked`

Air Transport Only

The parameter `"utilityOfLineSwitch"` is varied.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE config SYSTEM "http://www.matsim.org/files/dtd/config_v1.dtd"
  [
    <!ENTITY INPUTBASE "/net/ils3/dgrether/shared-svn/studies/countries/eu/
        flight/dg_oag_tuesday_flight_model_2_runways_3600vph_storage_restriction
        /">
    <!ENTITY OUTPUTBASE  "/net/ils3/dgrether/matsimOutput/run1876/">
  ]
>
<config>
  <module name="global" >
    <param name="coordinateSystem" value="EPSG:3395" />
    <param name="numberOfThreads" value="2" />
    <param name="randomSeed" value="4711" />
  </module>

  <module name="network" >
    <param name="inputNetworkFile" value="&INPUTBASE;air_network.xml" />
  </module>

  <module name="plans" >
    <param name="inputPlansFile" value="&INPUTBASE;../../../de/flight/demand/
        destatis/2009_september/population_september_2009_tabelle_2.2.2.xml.gz"
        />
  </module>

  <module name="scenario" >
    <param name="useTransit" value="true" />
    <param name="useVehicles" value="true" />
  </module>

  <module name="transit" >
    <param name="transitScheduleFile" value="&INPUTBASE;flight_transit_schedule.
        xml" />
    <param name="vehiclesFile" value="&INPUTBASE;flight_transit_vehicles.xml" />
  </module>

  <module name="controler" >
    <param name="eventsFileFormat" value="xml" />
    <param name="firstIteration" value="0" />
    <param name="lastIteration" value="600" />
    <param name="outputDirectory" value="&OUTPUTBASE;" />
```

```
  <param name="runId" value="1876" />
  <param name="writeEventsInterval" value="100" />
  <param name="mobsim" value="qsim" />
</module>

<module name="qsim" >
  <param name="flowCapacityFactor" value="1.0" />
  <param name="storageCapacityFactor" value="1.0" />
  <param name="removeStuckVehicles" value="false" />
  <param name="simStarttimeInterpretation" value="
      maxOfStarttimeAndEarliestActivityEnd" />
  <param name="snapshotStyle" value="queue" />
  <param name="stuckTime" value="100.0" />
  <param name="endTime" value="40:00:00" />
  <param name="numberOfThreads" value="1" />
</module>

<module name="otfvis" >
  <param name="linkWidth" value="50.0" />
  <param name="showTeleportation" value="false" />
</module>

<module name="planCalcScore" >
  <param name="BrainExpBeta" value="2.0" />
  <param name="performing" value="6.0" />
  <param name="traveling" value="-6.0" />
  <param name="travelingPt" value="-6.0" />
  <param name="lateArrival" value="-18.0" />
  <param name="travelingWalk" value="-6.0" />
  <param name="utilityOfLineSwitch" value="-0.0" />
  <param name="activityType_0" value="home" />
  <param name="activityTypicalDuration_0" value="21:00:00" />
</module>

<module name="strategy" >
  <param name="ModuleProbability_1" value="0.8" />
  <param name="ModuleProbability_2" value="0.1" />
  <param name="ModuleProbability_3" value="0.1" />
  <param name="Module_1" value="ChangeExpBeta" />
  <param name="Module_2" value="ReRoute" />
  <param name="ModuleDisableAfterIteration_2" value="500" />
  <param name="Module_3" value="TransitTimeAllocationMutator" />
  <param name="ModuleDisableAfterIteration_3" value="500" />
  <param name="maxAgentPlanMemorySize" value="5" />
</module>

<module name="TimeAllocationMutator">
  <param name="mutationRange" value="7200.0" />
</module>
```

```
  <module name="vspExperimental" >
    <param name="usingOpportunityCostOfTimeForPtRouting" value="true" />
    <param name="vspDefaultsCheckingLevel" value="warn" />
    <param name="isGeneratingBoardingDeniedEvent" value="true" />
  </module>
</config>
```

Alternative Mode

The parameter `"teleportedModeSpeed_train"` is varied.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE config SYSTEM "http://www.matsim.org/files/dtd/config_v1.dtd"
  [
    <!ENTITY INPUTBASE "/net/ils3/dgrether/shared-svn/studies/countries/eu/
        flight/dg_oag_tuesday_flight_model_2_runways_3600vph_storage_restriction
        /">
    <!ENTITY OUTPUTBASE  "/net/ils3/dgrether/matsimOutput/run1903/">
  ]
>
<config>
  <module name="global" >
    <param name="coordinateSystem" value="EPSG:3395" />
    <param name="numberOfThreads" value="2" />
    <param name="randomSeed" value="4711" />
  </module>

  <module name="network" >
    <param name="inputNetworkFile" value="&INPUTBASE;air_network.xml" />
  </module>

  <module name="plans" >
    <param name="inputPlansFile" value="&INPUTBASE;../../../de/flight/demand/
        destatis/2009_september/population_september_2009_tabelle_2.2.2.xml.gz"
        />
  </module>

  <module name="scenario" >
    <param name="useTransit" value="true" />
    <param name="useVehicles" value="true" />
  </module>

  <module name="transit" >
    <param name="transitScheduleFile" value="&INPUTBASE;flight_transit_schedule.
        xml" />
    <param name="vehiclesFile" value="&INPUTBASE;flight_transit_vehicles.xml" />
  </module>
```

```
<module name="controler" >
  <param name="eventsFileFormat" value="xml" />
  <param name="firstIteration" value="0" />
  <param name="lastIteration" value="600" />
  <param name="outputDirectory" value="&OUTPUTBASE;" />
  <param name="runId" value="1903" />
  <param name="writeEventsInterval" value="100" />
  <param name="mobsim" value="qsim" />
</module>

<module name="qsim" >
  <param name="flowCapacityFactor" value="1.0" />
  <param name="storageCapacityFactor" value="1.0" />
  <param name="removeStuckVehicles" value="false" />
  <param name="simStarttimeInterpretation" value="
      maxOfStarttimeAndEarliestActivityEnd" />
  <param name="snapshotStyle" value="queue" />
  <param name="stuckTime" value="100.0" />
  <param name="endTime" value="40:00:00" />
  <param name="numberOfThreads" value="1" />
</module>

<module name="otfvis" >
  <param name="linkWidth" value="50.0" />
  <param name="showTeleportation" value="false" />
</module>

<module name="planscalcroute" >
  <param name="beelineDistanceFactor" value="0.8" />
  <param name="networkModes" value="car" />
  <param name="teleportedModeSpeed_train" value="27.777777778" />
  <param name="teleportedModeFreespeedFactor_pt" value="2.0" />
  <param name="teleportedModeSpeed_bike" value="4.166666666666667" />
  <param name="teleportedModeSpeed_undefined" value="13.88888888888889" />
  <param name="teleportedModeSpeed_walk" value="0.8333333333333333" />
</module>

<module name="planCalcScore" >
  <param name="BrainExpBeta" value="2.0" />
  <param name="performing" value="6.0" />
  <param name="traveling" value="-6.0" />
  <param name="travelingPt" value="-6.0" />
  <param name="lateArrival" value="-18.0" />
  <param name="travelingWalk" value="-6.0" />
  <param name="travelingOther" value="-6.0" />
  <param name="utilityOfLineSwitch" value="-0.0" />
  <param name="activityType_0" value="home" />
  <param name="activityTypicalDuration_0" value="21:00:00" />
```

```
  </module>

  <module name="changeLegMode">
    <param name="modes" value="pt,train" />
    <param name="ignoreCarAvailability" value="true" />
  </module>

  <module name="strategy" >
    <param name="maxAgentPlanMemorySize" value="5" />
    <param name="ModuleProbability_1" value="0.7" />
    <param name="Module_1" value="ChangeExpBeta" />

    <param name="ModuleProbability_2" value="0.1" />
    <param name="Module_2" value="ReRoute" />
    <param name="ModuleDisableAfterIteration_2" value="500" />

    <param name="ModuleProbability_3" value="0.1" />
    <param name="ModuleDisableAfterIteration_3" value="500" />
    <param name="Module_3" value="TransitTimeAllocationMutator" />

    <param name="ModuleProbability_4" value="0.1" />
    <param name="Module_4" value="TransitChangeLegMode" />
    <param name="ModuleDisableAfterIteration_4" value="500" />
  </module>

  <module name="TimeAllocationMutator">
    <param name="mutationRange" value="7200.0" />
  </module>

  <module name="vspExperimental" >
    <param name="vspDefaultsCheckingLevel" value="warn" />
    <param name="isGeneratingBoardingDeniedEvent" value="true" />
    <param name="removingUnnecessaryPlanAttributes" value="true" />-->
  </module>
</config>
```

Path Size Logit for Plan Deletion

MATSim revision : r24284 (2013-05-24 18:09:32), playground revision: r25919, merged into r24284 manually to keep MATSim revision stable.

Executable: `playground.fuerbas.FlightControllerPSRemove`.

The parameter `"teleportedModeSpeed_train"` is varied.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE config SYSTEM "http://www.matsim.org/files/dtd/config_v1.dtd"
  [
```

```
    <!ENTITY INPUTBASE "/net/ils3/dgrether/shared-svn/studies/countries/eu/
        flight/dg_oag_tuesday_flight_model_2_runways_3600vph_storage_restriction
        /">
    <!ENTITY OUTPUTBASE  "/net/ils3/dgrether/matsimOutput/run1893/">
  ]
>
<config>
  <module name="global" >
    <param name="coordinateSystem" value="EPSG:3395" />
    <param name="numberOfThreads" value="2" />
    <param name="randomSeed" value="4711" />
  </module>

  <module name="network" >
    <param name="inputNetworkFile" value="&INPUTBASE;air_network.xml" />
  </module>

  <module name="plans" >
    <param name="inputPlansFile" value="&INPUTBASE;../../../de/flight/demand/
        destatis/2009_september/population_september_2009_tabelle_2.2.2.xml.gz"
        />
  </module>

  <module name="scenario" >
    <param name="useTransit" value="true" />
    <param name="useVehicles" value="true" />
  </module>

  <module name="transit" >
    <param name="transitScheduleFile" value="&INPUTBASE;flight_transit_schedule.
        xml" />
    <param name="vehiclesFile" value="&INPUTBASE;flight_transit_vehicles.xml" />
  </module>

  <module name="controler" >
    <param name="eventsFileFormat" value="xml" />
    <param name="firstIteration" value="0" />
    <param name="lastIteration" value="600" />
    <param name="outputDirectory" value="&OUTPUTBASE;" />
    <param name="runId" value="1893" />
    <param name="writeEventsInterval" value="100" />
    <param name="mobsim" value="qsim" />
  </module>

  <module name="qsim" >
    <param name="flowCapacityFactor" value="1.0" />
    <param name="storageCapacityFactor" value="1.0" />
    <param name="removeStuckVehicles" value="false" />
    <param name="simStarttimeInterpretation" value="
```

```
      maxOfStarttimeAndEarliestActivityEnd" />
  <param name="snapshotStyle" value="queue" />
  <param name="stuckTime" value="100.0" />
  <param name="endTime" value="40:00:00" />
  <param name="numberOfThreads" value="1" />
</module>


<module name="otfvis" >
  <param name="linkWidth" value="50.0" />
  <param name="showTeleportation" value="false" />
</module>

<module name="planscalcroute" >
  <param name="beelineDistanceFactor" value="0.8" />
  <param name="networkModes" value="car" />
  <param name="teleportedModeSpeed_train" value="27.777777778" />
  <param name="teleportedModeFreespeedFactor_pt" value="2.0" />
  <param name="teleportedModeSpeed_bike" value="4.166666666666667" />
  <param name="teleportedModeSpeed_undefined" value="13.88888888888889" />
  <param name="teleportedModeSpeed_walk" value="0.8333333333333333" />
</module>

<module name="planCalcScore" >
  <param name="PathSizeLogitBeta" value="60" />
  <param name="BrainExpBeta" value="2.0" />
  <param name="performing" value="6.0" />
  <param name="traveling" value="-6.0" />
  <param name="travelingPt" value="-6.0" />
  <param name="lateArrival" value="-18.0" />
  <param name="travelingWalk" value="-6.0" />
  <param name="travelingOther" value="-6.0" />
  <param name="utilityOfLineSwitch" value="-0.0" />
  <param name="activityType_0" value="home" />
  <param name="activityTypicalDuration_0" value="21:00:00" />
</module>

<module name="changeLegMode">
  <param name="modes" value="pt,train" />
  <param name="ignoreCarAvailability" value="true" />
</module>

<module name="strategy" >
  <param name="maxAgentPlanMemorySize" value="5" />
  <param name="ModuleProbability_1" value="0.7" />
  <param name="Module_1" value="ChangeExpBeta" />

  <param name="ModuleProbability_2" value="0.1" />
  <param name="Module_2" value="ReRoute" />
```

```
    <param name="ModuleDisableAfterIteration_2" value="500" />

    <param name="ModuleProbability_3" value="0.1" />
    <param name="ModuleDisableAfterIteration_3" value="500" />
    <param name="Module_3" value="TransitTimeAllocationMutator" />

    <param name="ModuleProbability_4" value="0.1" />
    <param name="Module_4" value="TransitChangeLegMode" />
    <param name="ModuleDisableAfterIteration_4" value="500" />
  </module>

  <module name="TimeAllocationMutator">
    <param name="mutationRange" value="7200.0" />
  </module>

  <module name="vspExperimental" >
    <param name="vspDefaultsCheckingLevel" value="warn" />
    <param name="isGeneratingBoardingDeniedEvent" value="true" />
    <param name="removingUnnecessaryPlanAttributes" value="true" />-->
  </module>
</config>
```

Appendix B

Airport Capacity

For selected Airports http://en.wikipedia.org/wiki/List_of_the_busiest_airports_in_Europe, 05.08.2012:

Num	Code	Dep	Arr	Total	Source
01	LHR	46.0	44.0	90.0	http://www.eurocontrol.int/sites/default/files/content/documents/nm/reports/network-operations-annual-report-2011-annex-02.pdf
02	CDG	63.0	54.0	109.0	http://www.cohor.org/
03	FRA	50.0	43.0	93.0	http://www.eurocontrol.int/sites/default/files/content/documents/nm/reports/network-operations-annual-report-2011-annex-02.pdf
04	AMS	74.0	68.0	112.0	http://www.eurocontrol.int/sites/default/files/content/documents/nm/reports/network-operations-annual-report-2011-annex-02.pdf
05	MAD	50.0	48.0	n/a	http://www.eurocontrol.int/sites/default/files/content/documents/nm/reports/network-operations-annual-report-2011-annex-02.pdf
06	MUC	58.0	58.0	90.0	http://www.eurocontrol.int/sites/default/files/content/documents/nm/reports/network-operations-annual-report-2011-annex-02.pdf
08	IST	28.0	28.0	50.0	http://www.eurocontrol.int/sites/default/files/content/documents/nm/reports/network-operations-annual-report-2011-annex-02.pdf

Num	Code	Dep	Arr	Total	Source
09	BCN	36.0	36.0	64.0	http://www.eurocontrol.int/sites/default/files/content/documents/nm/reports/network-operations-annual-report-2011-annex-02.pdf
10	LGW	28.0	27.0	52.0	http://www.acl-uk.org/UserFiles/File/Gatwick%20Summer%202012%20Capacity%20Declaration.pdf
11	ORY	24.0	27.0	n/a	http://www.cohor.org/
13	AYT	25.0	25.0	45.0	http://www.eurocontrol.int/sites/default/files/content/documents/nm/reports/network-operations-annual-report-2011-annex-02.pdf
14	ZRH	41.0	36.0	66.0	http://www.slotcoordination.ch/capacity.htm
15	PMI	33.0	33.0	62.0	http://www.eurocontrol.int/sites/default/files/content/documents/nm/reports/network-operations-annual-report-2011-annex-02.pdf
18	VIE	50.0	48.0	68.0	http://www.eurocontrol.int/sites/default/files/content/documents/nm/reports/network-operations-annual-report-2011-annex-02.pdf
20	DUS	36.0	33.0	47.0	https://sws.fhkd.org/EWPS/pAirportParameters.output?i_season=S12&i_button=RETRIEVE
22	ARN	42.0	42.0	84.0	http://www.arnslot.se/?name=declared
23	MAN	35.0	33.0	61.0	http://www.acl-uk.org/UserFiles/File/MAN%20Capacity%20declaration%20S12.pdf
24	BRU	40.0	35.0	45.0	http://www.brucoord.org/Page02.html
26	STN	28.0	28.0	50.0	http://www.acl-uk.org/UserFiles/File/STN_%2025-04-2012%20%20Declaration%20for%20Winter%202012.pdf
27	TXL	30.0	30.0	52.0	https://sws.fhkd.org/EWPS/pAirportParameters.output?i_season=S12&i_button=RETRIEVE
28	HEL	40.0	36.0	76.0	http://www.finavia.fi/vuosikertomukset/2004/en/hkivan_runway_capacity_inc.html
29	LIS	26.0	26.0	38.0	http://slotsportugal.ana.pt/en-US/main/airports/lisboa/capacity/Pages/default.aspx

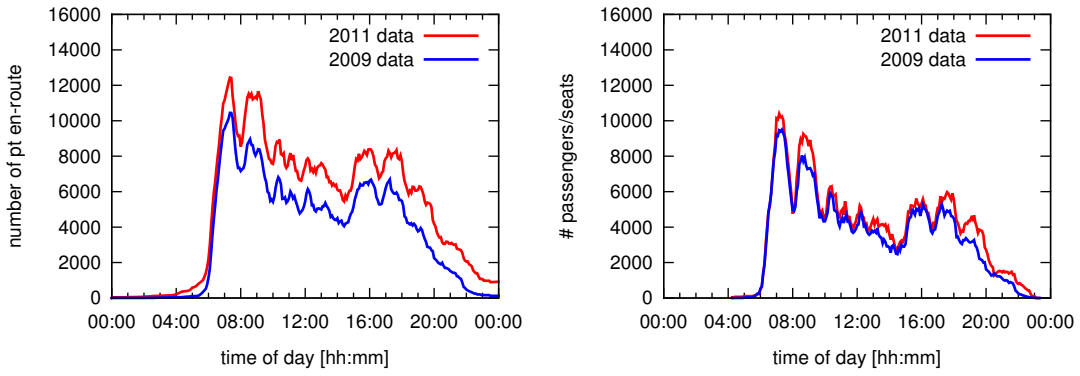
Num	Code	Dep	Arr	Total	Source
31	HAM	27.0	27.0	53.0	https://sws.fhkd.org/EWPS/pAirportParameters.output?i_season=S12&i_button=RETRIEVE
32	GVA	36.0	22.0	36.0	http://www.slotcoordination.ch/capacity.htm
34	SAW	8.0	8.0	20.0	http://www.eurocontrol.int/sites/default/files/content/documents/nm/reports/network-operations-annual-report-2011-annex-02.pdf
35	PRG	33.0	33.0	46.0	http://www.slot-czech.cz/en/site/capacity_parameters/summer-season-s12.htm
37	NCE	30.0	26.0	50.0	http://www.cohor.org/
39	CGN	40.0	40.0	52.0	https://sws.fhkd.org/EWPS/pAirportParameters.output?i_season=S12&i_button=RETRIEVE
41	STR	32.0	32.0	42.0	https://sws.fhkd.org/EWPS/pAirportParameters.output?i_season=S12&i_button=RETRIEVE
44	WAW	28.0	26.0	38.0	http://www.acl-uk.org/userFiles/File/WAW%20CAPACITY%20DECLARATION%20W12.pdf
46	BUD	30.0	26.0	n/a	http://www.hungarocontrol.hu/en/coordination-parameters

Appendix C

Passenger Demand for Air Transport Systems 2011 Data

The following tables and figures are derived from simulation runs for the air transport passenger demand model based on data from 09-2011. They are structurally equal to the results presented in Sec. 6.4 and not explained further. In contrast to the 2009 data, the synthetic population contains 65251 virtual persons, 1304 trips from the original data are neglected as origin and destination are equal. See Sec. 6.4 for definitions and interpretations.

C.1. Results 2009 vs 2011



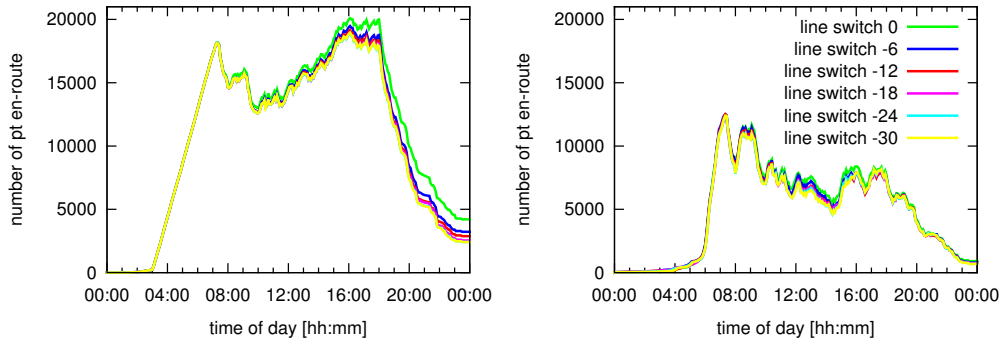
(a) 2009 vs 2011 passengers waiting for a flight or traveling by plane over time of day, $c_{lineswitch} = 0$, iteration 600

(b) 2009 vs 2011 passengers traveling by plane, $c_{lineswitch} = 0$, iteration 600

Figure C.1.: 2009 vs 2011 data

C.2. 2011 Data – No Random Selector for Plan Removal

C.2.1. No Alternative Mode



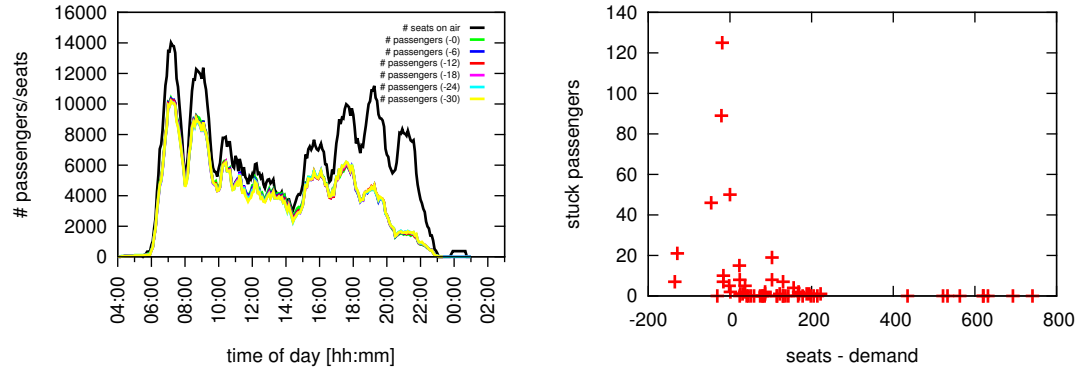
(a) After 0th iteration

(b) After 600th iteration

Figure C.2.: 09-2011 Data: Travelers en-route, i.e., waiting for a flight or traveling by plane, over time of day

$c_{lineswitch}$	σ^2	σ	mean rel error	stuck
$od_{transfer} - od_{direct}$	6715	82	1.56	-
-0	5248	72	0.55	1534
-6	5586	75	0.53	1469
-12	5713	76	0.65	1448
-18	5777	76	0.63	1480
-24	5785	76	0.62	1458
-30	5810	76	0.61	1456

Table C.1.: 09-2011 Data: Simulation results for different values of $c_{lineswitch}$



(a) Passengers and available seats over time within Germany

(b) Correlation between the available seats, the demand for seats and the number of passengers being stuck

Figure C.3.: 09-2011 Data: Potential reasons for stuck passengers

C.2.2. Adding an Alternative Mode

$v[km/h]$	σ^2	σ	mean rel error	stuck
$od_{transfer} - od_{direct}$	6715	82	1.56	-
100	5380	73	0.40	185
150	5605	75	0.39	69
200	6334	80	0.38	33
250	7580	87	0.43	29
300	11239	106	0.37	9

Table C.2.: 09-2011 Data: Simulation results for different values of v, simulation with alternative mode

$v[km/h]$	# air mode	# alt. mode	# stuck	air mode[%]	alt. mode[%]	stuck[%]
100	62726	2340	185	96.13	03.59	00.28
150	61379	3803	69	94.07	05.83	00.11
200	59491	5727	33	91.17	08.78	00.05
250	57248	7974	29	87.74	12.22	00.04
300	54089	11153	9	82.89	17.09	00.01

Table C.3.: 09-2011 Data: Modal split for different speeds of the alternative mode

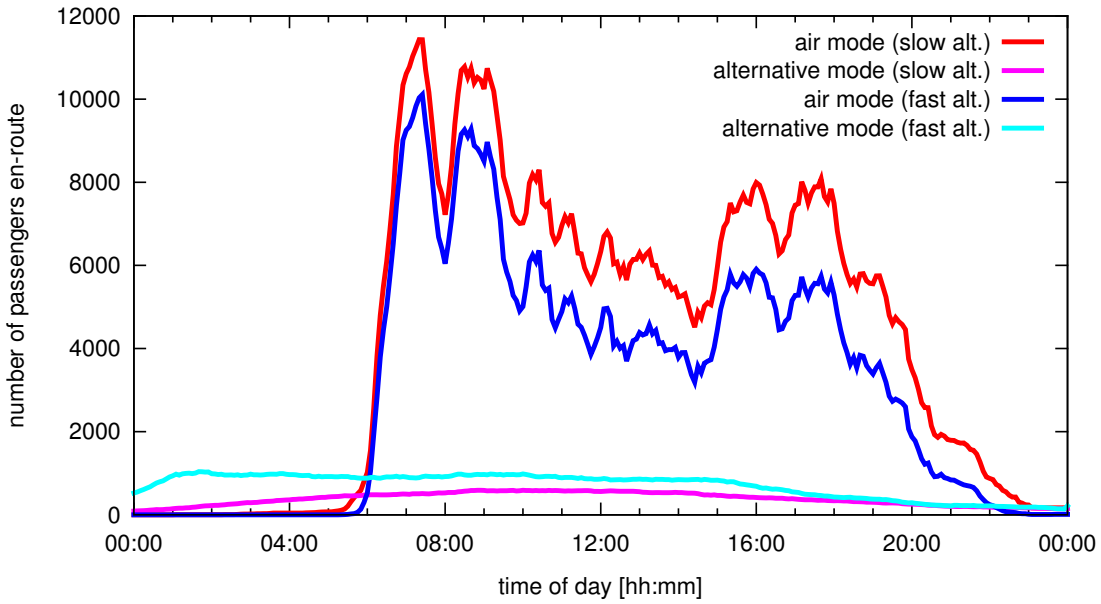


Figure C.4.: 09-2011 Data: Passengers waiting for a flight or traveling by plane or by the alternative mode over time of day

C.3. 2011 Data – Random Selector for Plan Removal

$v[km/h]$	σ^2	σ	mean rel error	stuck
$od_{transfer} - od_{direct}$	12640	112	1.75	-
100	3958	63	0.36	2
150	4273	65	0.37	1
200	7034	84	0.49	1
250	13013	114	0.59	1
300	23255	152	0.70	2

Table C.4.: 09-2011 Data & Random Selector for Plan Removal: Simulation results for different values of v

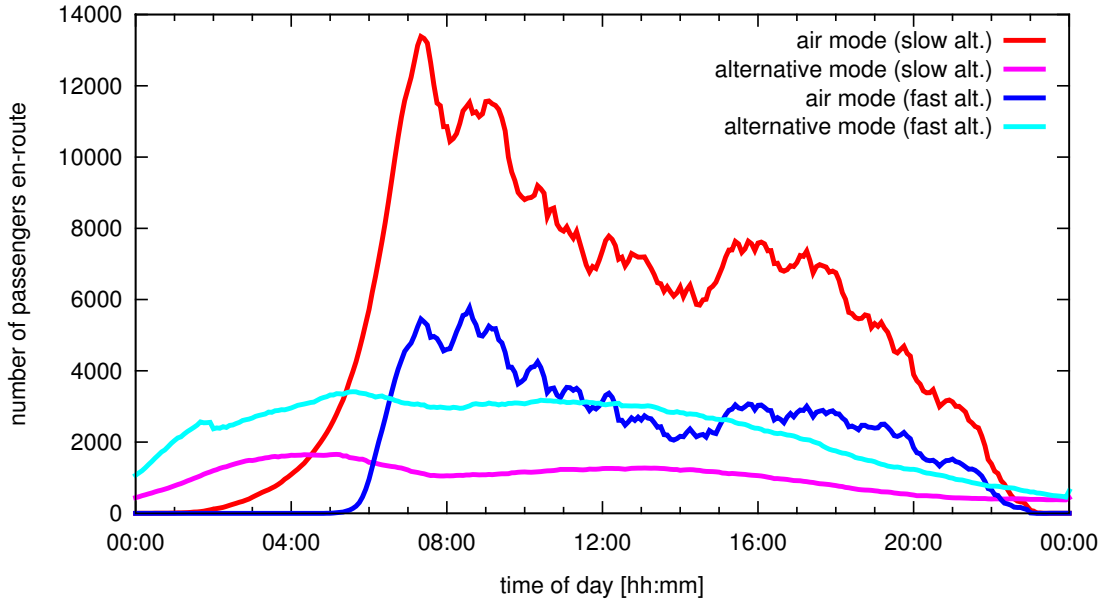


Figure C.5.: 09-2011 Data & Random Selector for Plan Removal: Passengers waiting for a flight or traveling by plane or by the alternative mode over time of day

$v[km/h]$	# air mode	# alt. mode	# stuck	air mode[%]	alt. mode[%]	stuck[%]
100	59647	5602	2	91.41	08.59	00.00
150	54220	11030	1	83.09	16.90	00.00
200	48213	17037	1	73.89	26.11	00.00
250	41280	23970	1	63.26	36.74	00.00
300	32709	32540	2	50.13	49.87	00.00

Table C.5.: 09-2011 Data & Random Selector for Plan Removal: Modal split for different speeds of the alternative mode