The Multi-Agent Transport Simulation MATSim

edited by

Andreas Horni, Kai Nagel, Kay W. Axhausen

Extract of selected chapters to serve as user guide.

— This version: May 2, 2022—

Contents

Title Page 1

Table of Contents i

I Using MATSim 1

1 Introducing MATSim 3
- 1.1 The Beginnings . . . 3
- 1.2 In Brief . . . 4
- 1.3 MATSim's Traffic Flow Model . . . 6
- 1.4 MATSim's Co-Evolutionary Algorithm . . . 7

2 Let's Get Started 9
- 2.1 Setting Up and Running MATSim . . . 9
  - 2.1.1 Setting Up MATSim . . . 9
  - 2.1.2 Running MATSim . . . 10
  - 2.1.3 Configuring MATSim . . . 11
- 2.2 An Example Scenario . . . 12
- 2.3 Building and Running a Basic Scenario . . . 14
  - 2.3.1 Typical Input Data . . . 14
  - 2.3.2 Typical Output Data . . . 17
  - 2.3.3 Units, Conventions, and Coordinate Systems . . . 18
  - 2.3.4 Data Requirements . . . 20
  - 2.3.5 Open Scenario Input Data . . . 21
- 2.4 MATSim Survival Guide . . . 22

3 Available Functionality and How to Use It 23
- 3.1 MATSim Modularity . . . 23
- 3.2 Levels of Access . . . 24
  - 3.2.1 Using MATSim through the GUI with config, network, and population only . . . 24
  - 3.2.2 Using MATSim through the GUI with additional files . . . 24
  - 3.2.3 Writing "Scripts in Java" . . . 24
  - 3.2.4 Using Contribs or External Extensions . . . 25
  - 3.2.5 Writing Your Own Extensions . . . 25
- 3.3 The Ideas Behind this Setup . . . 26
- 3.4 An Overview of Existing MATSim Functionality . . . 27

4 More About Configuring MATSim 31
- 4.1 General . . . 31
- 4.2 MATSim Data Containers . . . 31
  - 4.2.1 Network . . . 31

4.2.2 Population . . . 31
4.3 Global Modules and Global Aspects . . . 33
4.3.1 `controler` config file section . . . 33
4.3.2 Parallel Computing . . . 33
4.3.3 `global` config file section . . . 34
4.4 Mobility Simulations . . . 34
4.4.1 QSim . . . 35
4.4.2 JDEQSim . . . 36
4.5 Scoring . . . 36
4.6 Replanning Strategies . . . 36
4.6.1 Plans Generation and Removal (Choice Set Generation) . . . 37
4.6.2 Plan Selection (Choice) . . . 39
4.6.3 Innovation Switch-Off . . . 39
4.7 Observational Modules . . . 39
4.7.1 Travel Time Calculator . . . 39
4.7.2 Link Stats . . . 39
4.8 More Information . . . 40

5 A Closer Look at Scoring **41**
5.1 Good Plans and Bad Plans, Score and Utility . . . 41
5.2 The Current Charypar-Nagel Utility Function . . . 42
5.2.1 Mathematical Form . . . 42
5.2.2 Illustration . . . 46
5.2.3 The "Wrapping Around" of the Utility Function . . . 47
5.2.4 MATSim Scoring, Opportunity Cost of Time, and the VTTS . . . 47
5.2.5 The Resulting Modeling of Schedule Delay Costs . . . 48
5.3 Implementation Details . . . 49
5.3.1 Negative Durations . . . 49
5.3.2 Score Averaging . . . 50
5.3.3 Forcing Scores to Converge . . . 50
5.4 Typical Scoring Function Parameters and their Calibration . . . 51
5.5 Applications and Extensions of the Scoring Function . . . 52
5.6 Appendix: Old Version of the Zero Utility Duration . . . 53

II Extending MATSim **55**

6 Additional MATSim Data Containers **57**
6.1 Time-Dependent Network . . . 57
6.2 Person Attributes and Subpopulations . . . 57
6.3 Counts . . . 58
6.4 Facilities . . . 60
6.5 Households . . . 60
6.6 Vehicles . . . 60

7 Generation of the Initial MATSim Input **63**
7.1 Coordinate Transformations in Java . . . 63
7.2 Network Generation . . . 64
7.2.1 From OpenStreetMap . . . 64
7.2.2 From Other Sources . . . 64
7.3 Initial Demand Generation . . . 64
7.3.1 Simple Initial Demand . . . 64
7.3.2 Realistic Initial Demand . . . 65

11 QSim and its interaction with routing **67**
11.1 Other Modes than Car: Introduction . . . 67

11.2 Other Modes than Car: Routing Side . . . 68
11.2.1 Routing Side: Teleportation . . . 68
11.2.2 Routing Side: Network Modes . . . 69
11.2.3 Routing Side: Passenger Modes . . . 70
11.2.4 Other Routing Options . . . 70
11.3 Other Modes than Car: QSim Side . . . 70
11.3.1 QSim Side: Teleportation . . . 70
11.3.2 QSim Side: Network Modes . . . 71
11.3.3 QSim Side: Explicitly Simulated Passenger Modes . . . 71
11.3.4 QSim Side: Departure Handlers . . . 72
11.4 Other Modes than Car: Scoring Side . . . 73
11.5 Vehicle Types and Vehicles . . . 73
11.5.1 Vehicle sources and vehicle IDs . . . 73
11.5.2 Vehicle Placement and Behavior . . . 74
11.6 Other . . . 75

16 Modeling Public Transport with MATSim **77**
16.1 Basic Information . . . 77
16.2 Introduction . . . 77
16.3 Data Model and Simulation Features . . . 78
16.4 File formats . . . 79
16.4.1 `transitVehicles.xml` . . . 79
16.4.2 `transitSchedule.xml` . . . 79
16.4.3 Events for a public transit trip . . . 81
16.5 Swiss Rail Raptor . . . 82
16.6 Running with small sample sizes . . . 82
16.7 Possible Improvements . . . 82
16.8 Applications . . . 83

23 Dynamic Transport Services **85**
23.1 Introduction . . . 85
23.2 DVRP Contribution . . . 86
23.3 DVRP Model . . . 87
23.3.1 Schedule . . . 87
23.3.2 Least-Cost Paths . . . 88
23.4 DynAgent . . . 88
23.4.1 Main Interfaces and Classes . . . 88
23.4.2 Configuring and Running a Dynamic Simulation . . . 89
23.4.3 RandomDynAgent Example . . . 89
23.5 Agents in DVRP . . . 90
23.5.1 Drivers . . . 90
23.5.2 Passengers . . . 90
23.6 Optimizer . . . 91
23.7 Configuring and Running a DVRP Simulation . . . 92
23.8 OneTaxi Example . . . 92
23.9 Research with DVRP . . . 93

45 How to Write Your Own Extensions and Possibly Contribute Them to MATSim **95**
45.1 Introduction . . . 95
45.2 Preparation: Scripts-in-Java . . . 96
45.3 MATSim Extension Points: General . . . 97
45.4 Extension Points Related to Scenario . . . 97
45.4.1 Customizable, ObjectAttributes and Attributable . . . 97
45.4.2 Additional Scenario Elements . . . 98
45.5 Events . . . 98
45.6 Configuring the `Controler` (= inject syntax) . . . 99

45.6.1 Binding a Class (= an Implementation Type) . . . 100
45.6.2 Binding an Instance . . . 100
45.6.3 Binding a Provider (not important to get started) . . . 101
45.6.4 MATSim default bindings . . . 101
45.6.5 Advantages of the injection framework . . . 101
45.6.6 Pre-configured MATSim extension points . . . 102
45.6.7 ControlerListener: Handling Controler Events . . . 102
45.6.8 Mobsim Listener . . . 103
45.6.9 TripRouter . . . 103
45.6.10 Mobsim . . . 104
45.6.11 PlanStrategy . . . 104
45.6.12 Scoring . . . 105
45.7 Additional Config Groups . . . 106
45.8 MATSim extensions . . . 106
45.9 Conclusion . . . 106

44 Organization: Development Process, Code Structure and Contributing to MATSim 107
44.1 MATSim's Team, Core Developers Group, and Community . . . 107
44.2 Roles in the MATSim Community . . . 110
44.3 Code Base . . . 110
44.3.1 Main Distribution . . . 110
44.3.2 Core . . . 111
44.3.3 Contributions . . . 111
44.3.4 Playgrounds . . . 112
44.3.5 Contributions and Extensions . . . 113
44.3.6 Releases, Nightly Builds and Code HEAD . . . 113
44.4 Drivers, Organization and Tools of Development . . . 113
44.5 Documentation, Dissemination and Support . . . 114
44.6 Your Contribution to MATSim . . . 114

Acronyms 120

Glossary 125

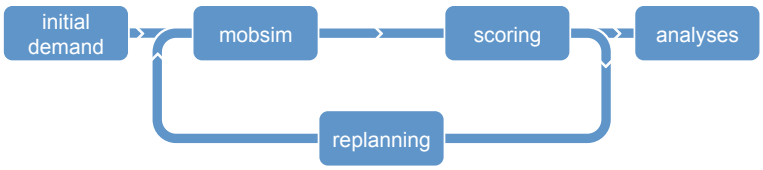
Bibliography 129

Part I
Using MATSim

1

Introducing MATSim

Authors: Andreas Horni, Kai Nagel, Kay W. Axhausen



1.1 The Beginnings

The MATSim (Multi-Agent Transport Simulation) project (MATSim, 2016) started with Kai Nagel, then at ETH Zürich, and his interest in improving his work with the TRANSIMS (TRansportation ANalysis and SIMulation System) project (Smith et al., 1995; FHWA, 2013); he also wanted to make the resulting code open-source.[1] After Kai Nagel's departure to Berlin in 2004, Kay W. Axhausen joined the team, bringing a different approach and experience. A collaboration, successful and productive for more than 10 years, was thus established, combining a physicist's and a civil engineer's perspective, as well as bringing together expertise in traffic flow, large-scale computation, choice modeling and CAS (Complex Adaptive Systems):

- **Microscopic modeling of traffic:** MATSim performs integral microscopic *simulation of resulting traffic flows* and the congestion they produce (see Section 1.3).
- **Microscopic behavioral modeling of demand/agent-based modeling:** MATSim uses a microscopic description of demand by *tracing the daily schedule* and the synthetic travelers' decisions. In retrospect, this can be called "agent-based".
- **Computational physics:** MATSim performs fast microscopic simulations with 10^7 or more "particles".

[1] TRANSIMS has, since then, also become open-source (TRANSIMS Open Source, 2013); but in 2000, it was difficult to procure in Europe.

- **Complex adaptive systems/co-evolutionary algorithms:** MATSim *optimizes the experienced utilities* of the whole schedule through the co-evolutionary search for the resulting equilibrium or steady state (see Section 1.4).

At the end of the 1990s, the scene was set for these research streams' mergence into a computationally efficient, modular, open-source software enabling further development on travel behavior, network response and efficient computation: MATSim.

1.2 In Brief

MATSim is an activity-based, extendable, multi-agent simulation framework implemented in Java. It is open-source and can be downloaded from the Internet (MATSim, 2016; GitHub, 2015). The framework is designed for large-scale scenarios, meaning that all models' features are stripped down to efficiently handle the targeted functionality; parallelization has also been very important (e.g., Dobler and Axhausen, 2011; Charypar, 2008). For the network loading simulation, for example, a queue-based model is implemented, omitting very complex and computationally expensive car-following behavior (see Section 1.3).

At this time, MATSim is designed to model a *single day*, the common unit of analysis for activity-based models (see, for example, the review by Bowman, 2009a). Nevertheless, in principle, a multi-day model could be implemented (Horni and Axhausen, 2012).

As shown in Section 1.4, MATSim is based on the co-evolutionary principle. Every agent repeatedly optimizes its daily activity schedule while in competition for space-time slots with all other agents on the transportation infrastructure. This is somewhat similar to the route assignment iterative cycle, but goes beyond route assignment by incorporating other choice dimensions like time choice (Balmer et al., 2005), mode choice (Grether et al., 2009), or destination choice (Horni et al., 2012) into the iterative loop.

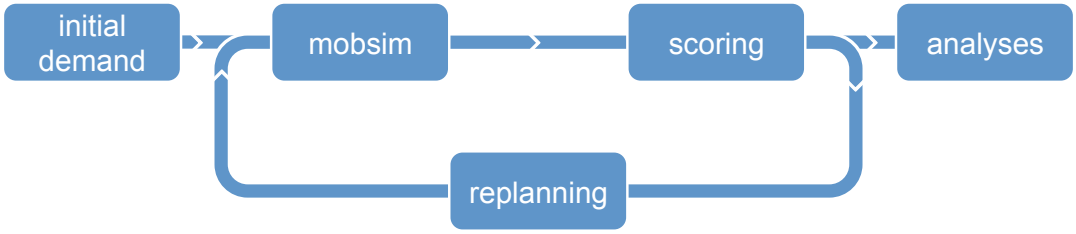


Figure 1.1: MATSim loop, sometimes called the MATSim cycle.

A MATSim run contains a configurable number of iterations, represented by the loop of Figure 1.1 and detailed below. It starts with an initial demand arising from the study area population's daily activity chains. The modeled persons are called agents in MATSim. Activity chains are usually derived from empirical data through sampling or discrete choice modeling. A variety of approaches is suitable, as evidenced in the scenarios' chapters (cf. Chapter **??**). During iterations, this initial demand is optimized individually by each agent. Every agent possesses a memory containing a fixed number of day plans,

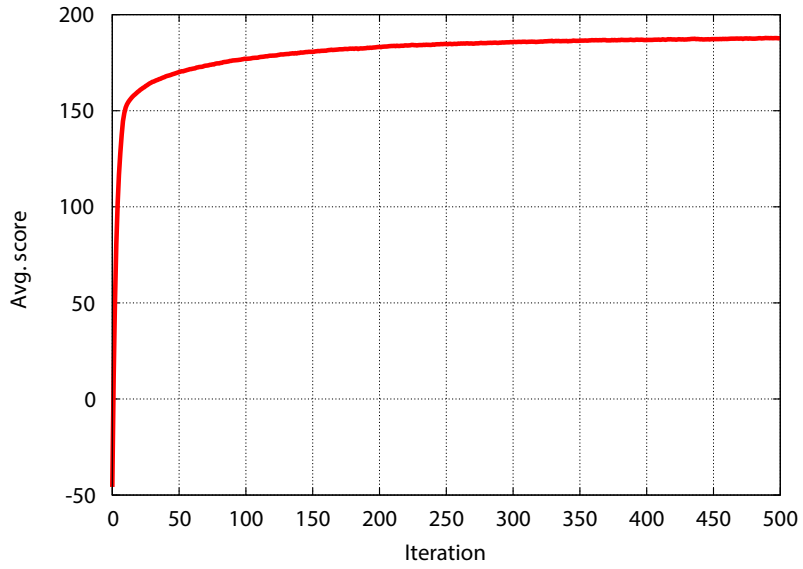


Figure 1.2: Typical score progress.

where each plan is composed of a daily activity chain and an associated score. The score can be interpreted as an econometric utility (cf. Chapter **??**).

In every iteration, prior to the simulation of the network loading with the MATSim *mobsim (mobility simulation)* (e.g., Cetin, 2005), each agent selects a plan from its memory. This selection is dependent on the plan *scores*, which are computed after each mobsim run, based on the executed plans' performances. A certain share of the agents (often 10 %) are allowed to clone the selected plan and modify this clone (*replanning*). For the network loading step, multiple mobsims are available and configurable (see Horni et al., 2011, and Section 4.4 of this book).

Plan modification is performed by the *replanning* modules. Four dimensions are usually considered for MATSim at this time: departure time (and, implicitly, activity duration) (Balmer et al., 2005), route (Lefebvre and Balmer, 2007), mode (Grether et al., 2009) and destination (Horni et al., 2009, 2012). Further dimensions, such as activity adding or dropping, or parking and group choices are currently under development and only available experimentally. MATSim replanning offers different strategies to adapt plans, ranging from random mutation to approximate suggestions, to best-response answers where, in every iteration, the currently optimal choice is searched. For example, routing often is a best-response modification, while time and mode replanning are random mutations.

Initial day chains do not have to be very carefully defined for the replanning dimensions included in the optimization process. Plausible values just speed up the optimization process.

If an agent ends up with too many plans (configurable), the plan with the lowest score (configurable) is removed from the agent's memory. Agents that have not undergone replanning select between existing plans. The selection model is configurable; in many MATSim investigations, a model generating a logit distribution for plan selection is used.

An iteration is completed by evaluating the agents' experiences with the selected day plans (*scoring*). The applied scoring function is described in detail in Chapter 5.

The iterative process is repeated until the average population score stabilizes. The typical score development curve (Figure 1.2, taken from Horni et al., 2009) takes the form of an evolutionary optimization progress (Eiben and Smith, 2003, Figure 2.5). Since the simulations are stochastic, one cannot use

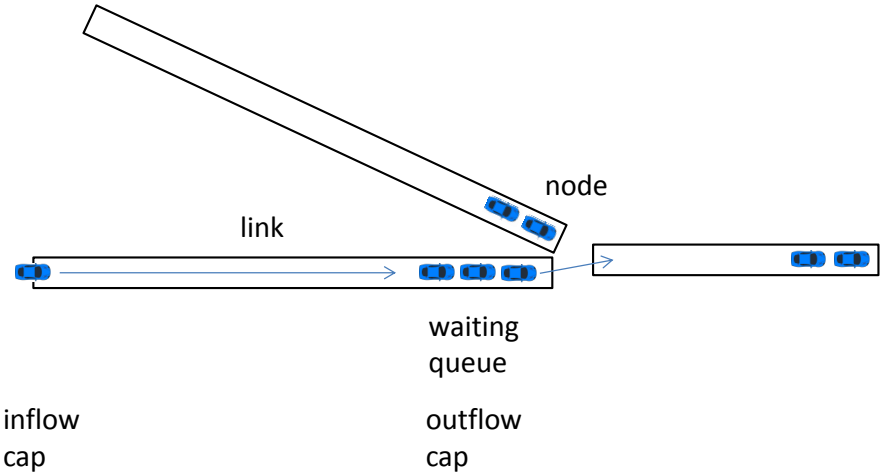


Figure 1.3: Traffic flow model.

convergence criteria appropriate for deterministic algorithms; for a discussion of possible approaches for the MATSim situation, see Sections **??** and **??** as well as Meister (2011).

MATSim offers considerable customizability through its modular design. Although implementing alternative core modules, such as an alternative network loading simulation, may entail substantial effort, in principle, every module of the framework can be exchanged. MATSim modules are described in Chapter 3 and following.

MATSim is strongly based on events stemming from the mobsim. Every action in the simulation generates an event, which is recorded for analysis. These event records can be aggregated to evaluate any measure at the desired resolution. The event architecture is detailed in Section 45.5.

1.3 MATSim's Traffic Flow Model

MATSim provides two internal mobsims: QSim and JDEQSim (Java Discrete Event Queue Simulation); in addition, external mobility simulations can be plugged in. Some years ago, the DEQSim (Discrete Event Queue Simulation) written in C++ and described by Charypar (2008); Charypar et al. (2007b,a, 2009) was plugged into MATSim and frequently used. The multi-threaded QSim is currently the default mobsim.

Charypar et al. (2009) distinguishes between

- physical simulations, featuring detailed car following models,
- cellular automata, in which roads are discretized into cells,
- queue-based simulations, where traffic dynamics are modeled with waiting queues,
- mesoscopic models, using aggregates to determine travel speeds, and
- macroscopic models, based on flows rather than single traveler units (e.g., cars).

As MATSim is designed for large-scale scenarios, it adopts the computationally efficient queue-based approach (see Figure 1.3). A car entering a network link (i.e., a road segment) from an intersection is added to the tail of the waiting queue. It remains there until the time for traveling the link with free flow has passed and until he or she is at the head of the waiting queue and until the next link allows entering. The approach is very efficient, but clearly it comes at the price of reduced resolution, i.e., car following effects are not captured. In JDEQSim, for computational reasons, the waiting-queue approach is combined with an event-based update step (Charypar et al., 2009). In other words, there

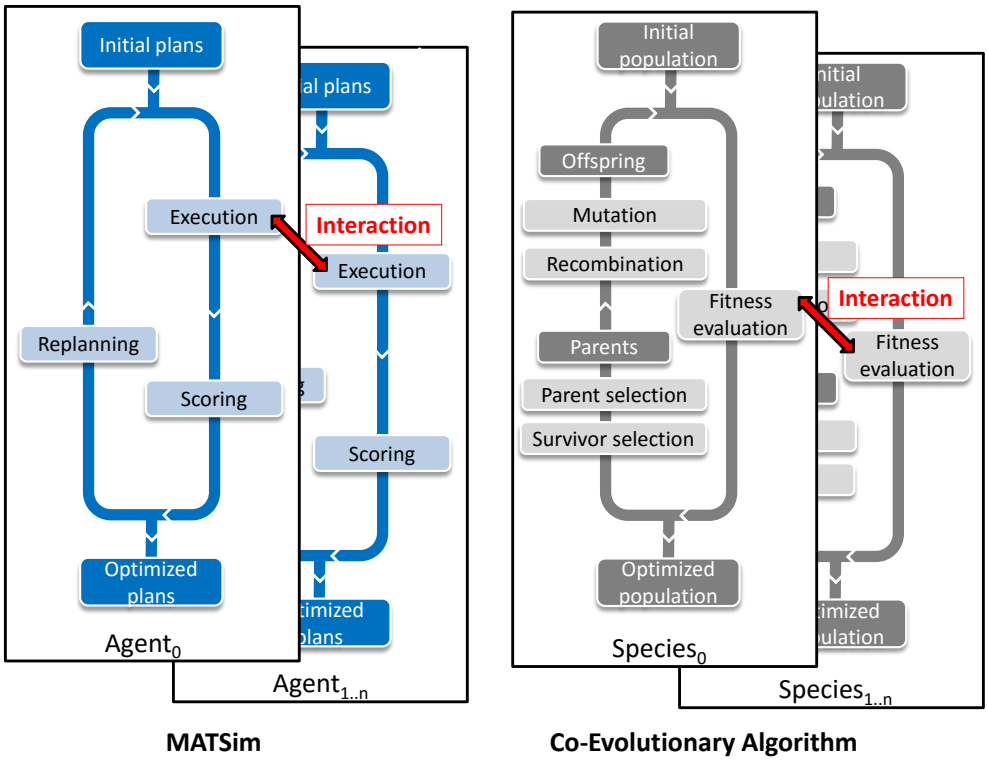


Figure 1.4: The co-evolutionary algorithm in MATSim.

is no time-step-based updating process of any agent in the scenario. Instead agents are only touched if they actually require an action. For example, links do not have to be processed while agents traverse them. Update events triggering is managed by a global scheduler. QSim, however, is time-step based. The MATSim traffic flow model is strongly based on the two link attributes: storage capacity and flow capacity. Storage capacity defines the number of cars fitting onto a network link.

Flow capacity specifies the outflow capacity of a link, i.e., how many travelers can leave the respective link per time step. It is an individual attribute of the link. The current implementation of QSim has no *maximum* inflow capacity specified. In contrast, in the earlier DEQSim and current JDEQSim, an inflow capacity can also be specified, which may move jams at merges from the end of the first common link, where the QSim generates them, upstream to where the links merge and where they plausibly should be (Charypar, 2008, p. 99). However, additional data is needed for this, which is often not available.

This basic traffic flow model has been extended with various modules: Signals and multiple lane modeling are available for the QSim (Chapter **??**); backward-moving gaps, as investigated by Charypar (2008), are included in both in QSim (see Section 4.4.1) and in JDEQSim. Interactions between different modes are described in Section 11.1 and Chapter **??**.

1.4 MATSim's Co-Evolutionary Algorithm

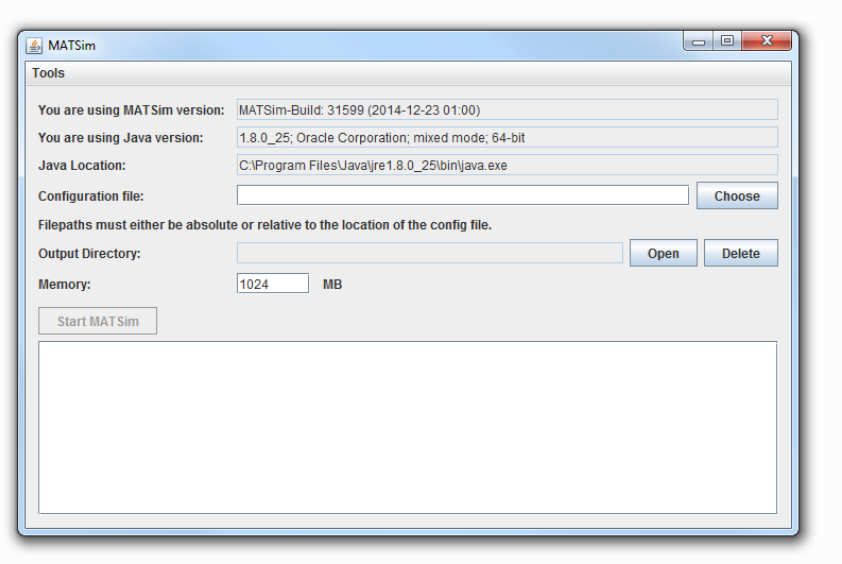
As illustrated in Figure 1.4, the MATSim equilibrium is searched for by a *co-evolutionary algorithm* (see, e.g., Popovici et al., 2012). These algorithms co-evolve different species subject to interaction (e.g., competition). In MATSim, individuals are represented by their plans, where a person represents a species. With the co-evolutionary algorithm, optimization is performed in terms of agents' plans, i.e., across the whole daily plan of activities and travel. It achieves more than the standard traffic flow equilibria, which ignores activities. Eventually, an equilibrium is reached, subject to constraints, where the agents cannot further improve their plans unilaterally.

Note that there is a difference between the application of an evolutionary algorithm and a *co*-evolutionary algorithm. An evolutionary algorithm would lead to a system optimum, as optimization is applied with a global (or population) fitness function. Instead, the co-evolutionary algorithm leads to a (stochastic) user equilibrium, as optimization is performed in terms of *individual* scoring functions and within an agent's set of plans.

2

Let's Get Started

Authors: Marcel Rieser, Andreas Horni, Kai Nagel



This chapter explains how to set up and run MATSim and describes the requirements for building a basic scenario. Updated information may be available from `http://matsim.org`, in particular from `http://matsim.org/docs`.

Getting the source code into different computing environments and extending MATSim through the API (Application Programming Interface) is described in Part II, Chapter 45.

2.1 Setting Up and Running MATSim

2.1.1 Setting Up MATSim

To run MATSim, you must install the Java SE (Java Standard Edition) that complies with the appropriate MATSim version. At this time, this is Java SE 11.

MATSim example project on github You can fork or clone a so-called example project on GitHub, see `https://github.com/matsim-org/matsim-example-project`.

This version is targeted to programmers who are fluent with an IDE (Integrated Development Environment) (e.g. Eclipse, IntelliJ) and Java, and who want program against MATSim. The approach will automatically download MATSim (as a so-called Maven articfact), allow you to browse the source code, and keep you up-to-date with releases or snapshots. It will not allow you to modify the existing MATSim code—which, in most cases, also should not be necessary: it is preferred that you contact the developers in such situations and we will try to help or implement missing extension points.

Many Java programs, along the lines discussed in Section 3.2.3, are provided in the so-called code examples project on GitHub, see `https://github.com/matsim-org/matsim-code-examples`.

Standalone The "Standalone" version is targeted to users who are not fluent with an IDE (e.g. Eclipse) and Java, and want to use MATSim by editing the input files, including `config.xml`. A basic GUI (Graphical User Interface) is provided.

You will need the official *MATSim release,* a zip file (usually designated with the version number `matsim-yy.yy.yy.zip`), that includes everything required to run it. It can be downloaded following the corresponding links under `http://matsim.org/downloads`. Unzip results in the **MATSim directory tree**.

Maven One can use MATSim as an Maven plugin; both release versions and snapshots are available. See again `http://matsim.org/downloads` for more information.

Again, some Java programs, along the lines discussed in Section 3.2.3, are provided in the so-called code examples project on GitHub, see `https://github.com/matsim-org/matsim-code-examples`.

Browsing the source code If you just want to look at code without downloading and installing a zip file: On GitHub, the root of the **MATSim directory tree** is at `https://github.com/matsim-org/matsim-libs`.

Alternatively, if you have installed the MATSim example project with Maven, then the MATSim sources are automatically available inside the IDE.

Other options `http://matsim.org/downloads` describes additional options, including how to obtain older or newer versions or how to add extensions.

2.1.2 Running MATSim

GUI from jar file Starting with version 0.8.x, MATSim can be started by double-clicking the MATSim JAR (Java ARchive) file. A minimal GUI, as shown in Figure 2.1, opens and the MATSim run can be

1. configured—by choosing a configuration file as indicated by the "Choose" button—, and
2. started—by clicking on "Start MATSim".

If the output directory (as defined in the config file, see below) is already there, it needs to be deleted before the run can be successfully started; there is "Delete" button to achieve this.

After the run has successfully finished, the output directory can be opened in a file browser with the "Open" button. Output files, such as `output_network.xml.gz` and `output_events.xml.gz`, can be dragged-and-dropped into the Via visualization software, and simulated traffic can be played back.[1]

GUI from IDE Alternatively, you can run the `MATSimGUI` class from the IDE.

[1] Via is commercial, see `https://www.simunto.com/via`. A free license is available for up to 500 MATSim agents. An alternative is OTFVis (On The Fly Visualizer) (Chapter **??**), but it is not officially supported, and more difficult to use.

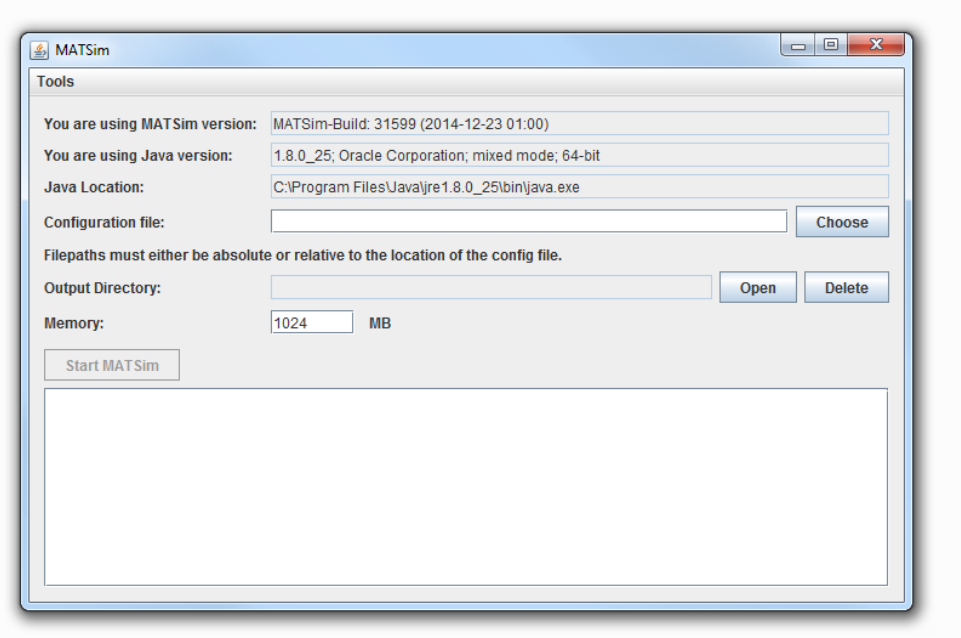


Figure 2.1: Minimal MATSim GUI.

Without GUI from IDE `MATSimGUI` points to `RunMatsim`. One can instead just run `RunMatsim`. Sorting out path names is a bit more challenging, but it is more convenient in the long run.

2.1.3 Configuring MATSim

MATSim is configured in the config file. It builds the connection between the user and MATSim and containing a list of settings that influence how the simulation behaves.

All configuration parameters are simple pairs of a parameter `name` and a parameter `value`. The parameters are grouped into logical groups; one group has settings related to the `Controler`, like the number of iterations, or another group has settings for the mobsim, e.g., end time of the mobsim. As shown in Chapter 3, numerous MATSim modules can be added to MATSim and configured by specifying the respective configuration file section.

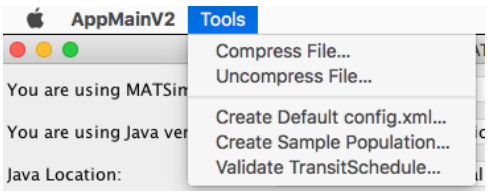


Figure 2.2: MATSim GUI tools.

The list of available parameters and valid parameter values may vary from release to release. For a list of all available settings available with the version you are working with, run the "Create Default config.xml" MATSim GUI tool, see Fig. 2.2.[2] This will create a new config file, containing all available parameters, along with their default values and often an explanatory comment, making it easier to see what settings are available. To use and modify specific settings, lines with their corresponding

[2] Outside the MATSim GUI, this can be achieved by running the following command from the command line:

```
java -cp /path/to/matsim.jar org.matsim.run.CreateFullConfig fullConfig.xml
```

parameters can be copied to the config file, specific to the scenario to be simulated, and the parameter values can be modified in that file.

A fairly minimal config file contains the following information:

```
<module name="network">
   <param name="inputNetworkFile" value="<path-to-network-file>" />
</module>

<module name="plans">
   <param name="inputPlansFile" value="<path-to-plans-file>" />
</module>

<module name="controler">
   <param name="firstIteration" value="0" />
   <param name="lastIteration" value="0" />
</module>

<module name="planCalcScore" >
   <parameterset type="activityParams" >
      <param name="activityType" value="h" />
      <param name="typicalDuration" value="12:00:00" />
   </parameterset>
   <parameterset type="activityParams" >
      <param name="activityType" value="w" />
      <param name="typicalDuration" value="08:00:00" />
   </parameterset>
</module>
```

For a working example, check https://github.com/matsim-org/matsim-code-examples/blob/14.x/scenarios/equil/config.xml or https://github.com/matsim-org/matsim-example-project/tree/master/scenarios/equil/config.xml. Note that since version 0.9.x, most file names are relative to the directory of the config file. Input file names can also be given as URLs (Uniform Resource Locators).

In the example, supply is provided by the network and demand by the plans file. Typical input data is described in Section 2.3.1. The specification that the first and last iteration are the same means that no replanning of the demand is performed. What *is* executed is the mobsim (Figure 1.1), followed by each executed plan's performance scoring. To function, the scoring needs to know, from the config file, all activity types used in the plans and the typical duration for each activity type.

Further configuration possibilities are described in Chapter 4.

2.2 An Example Scenario

The MATSim release is shipped with an example scenario named `equil` in the folder `examples/scenarios/equil`,[3] containing these files: `config.xml`, `network.xml`, `plans100.xml`, and `plans2000.xml.gz`, containing, respectively, 100 and 2000 persons with their daily plans, using car mode only. A tiny population containing only 2 persons (`plans2.xml`), one using public transport, the other using car mode, is also provided. An example for count data is also found in the folder (`counts100.xml`).

In addition, there is also a file with 100 *trips* (`plans100trips.xml`), i.e., demand going only from one location to another, using a `dummy` activity type at each end. This is provided to show that MATSim can also be run as a fully trip-based approach, without considering any activities. Clearly, it loses some of its expressiveness, but the basic concepts, including route and even departure time adaptation, still work in exactly the same way.

The scenario network is shown in Figure 2.3.

The following lines explain the scenario by discussing the most important sections from the config file `config.xml`.

[3] See https://github.com/matsim-org/matsim-code-examples/tree/14.x/scenarios/equil or https://github.com/matsim-org/matsim-example-project/tree/master/scenarios/equil.

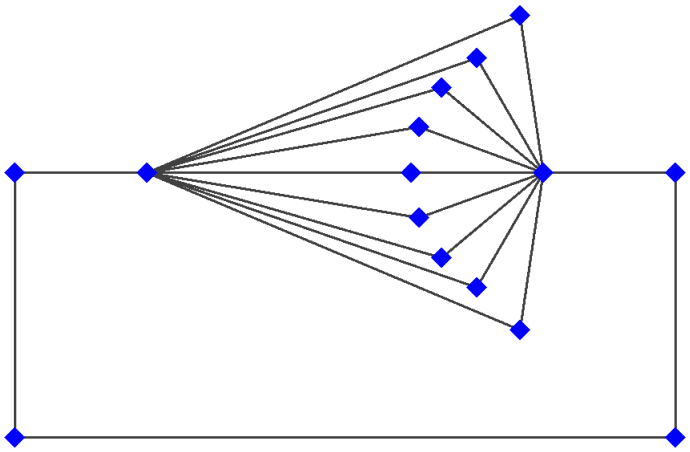


Figure 2.3: Equil scenario network.

"strategy" section of the config file As shown in the config file excerpt below, this scenario uses replanning. In each iteration, 10 % of the agents reroute one of their plans (module `ReRoute`). The remaining 90 % select their highest score plan for re-execution in the current iteration (module `BestScore`). Plans are deleted from the agent's memory if it is full, defined by `maxAgentPlanMemorySize`. By default, the plan with the lowest score is removed; this is configurable and currently being researched (see Section 44.6).

```
<module name="strategy">
   <param name="maxAgentPlanMemorySize" value="5" /> <!-- 0 means unlimited -->

   <parameterset type="strategysettings" >
      <param name="strategyName" value="ReRoute" />
      <param name="weight" value="0.1" />
   </parameterset>

   <parameterset type="strategysettings" >
      <param name="strategyName" value="BestScore" />
      <param name="weight" value="0.9" />
   </parameterset>

</module>
```

"planCalcScore" section of the config file The section `planCalcScore` defines parameters used for scoring, explained in Chapter 5. As seen in the example, two activity types, `h` (home) and `w` (work), are specified. All activity types contained in the population file (cf. Section 2.3.1.2) must be defined in the `planCalcScore` section of the config file.

```
<module name="planCalcScore" >
   <parameterset type="activityParams" >
      <param name="activityType" value="h" />
      <param name="typicalDuration" value="12:00:00" />
   </parameterset>
   <parameterset type="activityParams" >
      <param name="activityType" value="w" />
      <param name="typicalDuration" value="08:00:00" />
   </parameterset>
</module>
```

"controler" section of the config file The scenario is run for 10 iterations, writes the output files to `./output/equil` (Section 2.3.2) and uses QSim as the mobsim (more on mobsims in Sections 1.3, 4.4 and 11).

```
<module name="controler">
   <param name="outputDirectory" value="./output/equil" />
   <param name="lastIteration" value="10" />
   <param name="mobsim" value="qsim" />
</module>
```

Visualization Simulation results can be visualized with Via (Chapter **??**) or OTFVis (Chapter **??**).

2.3 Building and Running a Basic Scenario

This section provides information on typical input data files used for a MATSim experiment, as well as the standard output files generated. The section starts with typical input and output data. It then moves on to units, conventions and coordinate systems used in MATSim, followed by hints on practical data requirements. The section is concluded by pointers to open, i.e. freely available scenario input data.

2.3.1 Typical Input Data

Minimally, MATSim needs the files

- `config.xml`, containing the configuration options for MATSim and presented above in Section 2.1.3,
- `network.xml`, with the description of the (road) network, and
- `population.xml`, providing information about travel demand, i.e., list of agents and their daily plans.

Thus, `population.xml` and `network.xml` might get quite large. To save disk space, MATSim supports reading and writing data in a compressed format. MATSim uses GZIP-compression for this. Thus, many file names have the additional suffix `.gz`, as in `population.xml.gz`. MATSim recognizes whether files are compressed, or should be written compressed, based on file name.

In more detail, the network and population files resemble the following; for the config file, see Section 2.1.3 above.

2.3.1.1 Network file

Network is the infrastructure on which agents (or vehicles) can move around. The network consists of nodes and links (in graph theory, also called vertices and edges). A simple network description in MATSim's XML (Extensible Markup Language) data format could contain approximately the following information:

```
<network name="example network">
   <nodes>
      <node id="1" x="0.0" y="0.0"/>
      <node id="2" x="1000.0" y="0.0"/>
      <node id="3" x="1000.0" y="1000.0"/>
   </nodes>
   <links>
      <link id="1" from="1" to="2" length="3000.00" capacity="3600"
            freespeed="27.78" permlanes="2" modes="car" />
      <link id="2" from="2" to="3" length="4000.00" capacity="1800"
            freespeed="27.78" permlanes="1" modes="car" />
      <link id="3" from="3" to="2" length="4000.00" capacity="1800"
            freespeed="27.78" permlanes="1" modes="car" />
      <link id="4" from="3" to="1" length="6000.00" capacity="3600"
```

```
            freespeed="27.78" permlanes="2" modes="car" />
    </links>
</network>
```

For a working example, see under https://github.com/matsim-org/matsim-code-examples/tree/14.x/scenarios/equil or https://github.com/matsim-org/matsim-example-project/tree/master/scenarios/equil.

Each element has an identifier `id`. Nodes are described by an `x` and a `y` coordinate value (also see Sections 2.3.3.3 and 7.1). Links have more features; the `from` and `to` attributes reference nodes and describe network geometry. Additional attributes describe traffic-related link aspects:

- The `length` of the link, typically in meters (see Section 2.3.3).
- The flow `capacity` of the link, i.e., number of vehicles that traverse the link, typically in vehicles per hour.
- The `freespeed` is the maximum speed that vehicles are allowed to travel along the link, typically in meters per second.
- The number of lanes (`permlanes`) available in the direction specified by the 'from' and 'to' nodes.
- The list of `modes` allowed on the link. This is a comma-separated list, e.g., `modes="car, bike, taxi"`.

All links are uni-directional. If a road can be traveled in both directions, two links must be defined with reversed `to` and `from` attributes (see links with id 2 and 3 in the listing above).

2.3.1.2 Population file = plans file

File Format MATSim travel demand is described by the agents' daily plans. The full set of agents is also called the population, hence the file name `population.xml`. Alternatively, `plans.xml` is also commonly used in MATSim, as the population file essentially contains a list of daily plans.

The population contains the data in a hierarchical structure, as shown in the following example. This example illustrates the data structure; minimal input files need less information, as illustrated later.

```
<population>
    <person id="1">
        <plan selected="yes" score="93.2987721">
            <act type="home" link="1" end_time="07:16:23" />
            <leg mode="car">
                <route type="links">1 2 3</route>
            </leg>
            <act type="work" link="3" end_time="17:38:34" />
            <leg mode="car">
                <route type="links">3 1</route>
            </leg>
            <act type="home" link="1" />
        </plan>
    </person>
    <person id="2">
        <plan selected="yes" score="144.39002">
            ...
        </plan>
    </person>
</population>
```

For a working example, see under https://github.com/matsim-org/matsim-code-examples/tree/14.x/scenarios/equil or https://github.com/matsim-org/matsim-example-project/tree/master/scenarios/equil.

The population contains a list of persons, each person contains a list of plans, and each plan contains a list of activities and legs.

Exactly one plan per person is marked as selected. Each agent's selected plan is executed by the mobility simulation. During the replanning stage, a different plan might become selected. A plan can contain a score as attribute. The score is calculated and stored in the plan after its execution by the mobility simulation during the scoring stage.

The list of activities and legs in each plan describe each agent's planned actions. Activities are assigned a type and typically have—except for the last activity in a daily plan—a defined end time. There are some exceptions where activities have a duration instead of an end time. Such activities are often automatically generated by routing algorithms and are not described in this book. To describe the location where an activity takes place, the activity is either assigned a coordinate by giving it an `x` and `y` attribute value, or it has a link assigned, describing from which link the activity can be reached. Because the simulation requires a link attribute, `Controler` calculates the nearest link for a given coordinate when the link attribute is missing.

A leg describes how an agent plans to travel from one location to the next; each leg must have a transport mode assigned. Optionally, legs may have an attribute, `trav_time`, describing the expected travel time for the leg. For a leg to be simulated, it must contain a route. The format of a route depends on the mode of a leg. For car legs, the route lists the links the agent has to traverse in the given order, while for transit legs, information about stop locations and expected transit services are stored. MATSim automatically computes initial routes for initial plans that do not contain them.

An agent starts a leg directly after the previous activity (or leg) has ended. The handling of the agent in the mobsim depends on the mode. By default, car and transit legs are well-supported by the mobsim. If the mobsim encounters a mode it does not know, it defaults to teleportation. In this case, an agent is removed from the simulated reality and re-inserted at its target location after the leg's expected travel time has passed.

A Minimal Population File The population data format is one of the most central data structures in MATSim and might appear a bit overwhelming at first. Luckily, to get started, it is only necessary to know a small subset. A population file needs, approximately, only the following information:

```
<population>
   <person id="1">
      <plan>
         <act type="home" x="5.0" y="8.0" end_time="08:00:00" />
         <leg mode="car" />
         <act type="work" x="1500.0" y="890.0" end_time="17:30:00" />
         <leg mode="car" />
         <act type="home" x="5.0" y="8.0" />
      </plan>
   </person>
   <person id="2">
      ...
   </person>
</population>
```

See plans-minimal.xml in matsim-code-examples for a working example.

The following items can be used for simplification:

- Each person needs not more than one plan.
- The plan does not have to be selected or have a score.
- Activities can be located just by their coordinates.
- Activities should have a somewhat reasonable end-time.
- Legs need only a mode, no routes.

When a simulation is started, MATSim's `Controler` will load such a file and then automatically assign the link nearest to each activity and calculate a suitable route for each leg. This makes it easy to get started.

2.3.2 Typical Output Data

MATSim creates output data that can be used to analyze results as well as to monitor the current simulation setup progress. Some of the files summarize a complete MATSim run, while others are created for a specific iteration only. The first type of files goes directly to the output folder's top level, which can be specified in the `controler` section of the config file. The other files are stored in iteration-specific folders `ITERS/it.{iteration number}`, which are continuously created in the output folder. For some files (typically for large ones, such as population), the output frequency can be specified in the config file. They then go only to the respective iteration folders. The files summarizing the complete MATSim run are built 'on the fly', i.e., after every iteration, currently computed iteration values are stored, allowing continuous monitoring of the run. Some files are created by default (such as the score statistics files); others need to be triggered by a respective configuration file section (such as count data files).

The following output files are continuously built up to summarize the complete run.

Log File: During a MATSim run, a log file is printed containing information you might need later for your analyses, or in case a run has crashed.

Warnings and Errors Log File: Sometimes, MATSim identifies problems in the simulation or its configuration; it will then write warning and error messages to the log file. Because the log file contains so much information, these warnings can be overlooked. For this reason, a separate log file is generated in the run output directory, containing only warnings and error messages. It is important to check this file during/after a run for possible problems.

Score Statistics: Score statistics are available as a picture (`scorestats.png`), as well as a text file (`scorestats.txt`). They show the average best, worst, executed and overall average of all agents' plans for every iteration. An example score plot is shown in Figure 1.2.

Leg Travel Distance Statistics: Leg travel distance statistics (files `traveldistancestats.png` and `traveldistancestats.txt`) are comparable to score statistics, but instead, they plot travel distance.

Stopwatch: The stopwatch file (`stopwatch.txt`) contains the computer time (so-called wall clock time) of actions like replanning or the execution of the mobsim for every iteration. This data is helpful for computational performance analyses, e.g., how long does replanning take compared to the mobility simulation?

The following output files are created for specific iterations:

Events: Every action in the simulation is recorded as a MATSim event, be it an activity start or change of network link; see Fig. 2.4. Each event possesses one or multiple attributes. By default, the time when the event occurred is included. Additionally, information like the ID of the agent triggering the event, or the link ID where the event occurred, could be included. The events file is an important base for post-analyses, like the visualizers. Events are discussed in detail in Section 45.5.

Plans: At configurable iterations, the current state of the population, with the agents' plans, is printed. The final iteration's plans are also generated on the top level of the output folder.

Leg Histogram: In every iteration, a leg histogram is plotted. A leg histogram depicts the number of agents arriving, departing or en route, per time unit. Histograms are created for each transport mode and for the sum of all transport modes. Each file starts with the iteration number and ends

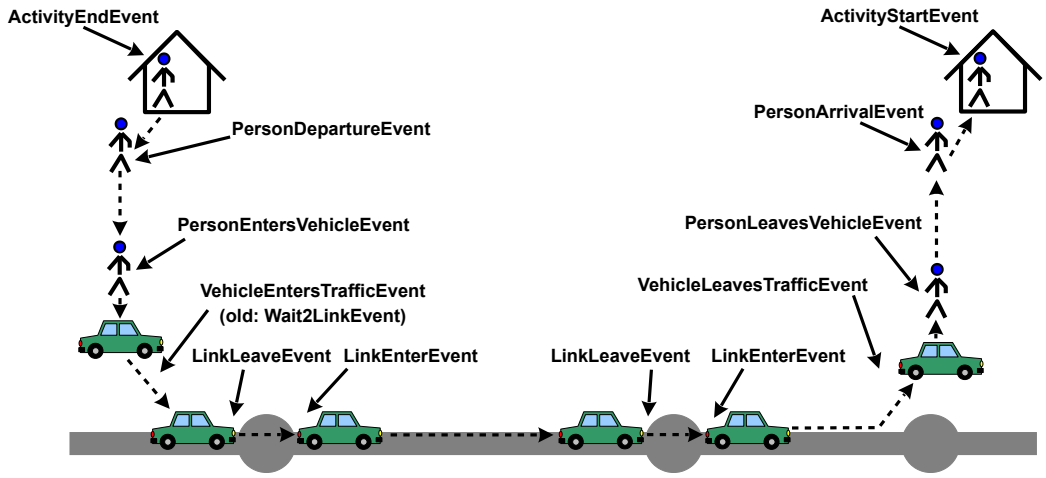


Figure 2.4: Mobsim events.

with the transport mode (e.g., `1.legHistogram_car.png` or `1.legHistogram_all.png`). A text file is also created (e.g., `1.legHistogram.txt`), containing the data for all transport modes.

Trip Durations: For each iteration, a trip durations text file (e.g., `1.tripdurations.txt`), listing number of trips and their durations, on a time bin level for each activity pair (e.g., from work to home or from home to shopping), is produced.

Link Stats: In each iteration, a link stats file containing hourly count values and travel times on every network link is printed. Link stats are particularly important for comparison with real-world count data, as introduced in Section 6.3.

2.3.3 Units, Conventions, and Coordinate Systems

2.3.3.1 Units

MATSim tries to make few assumptions about actual units, but it is sometimes necessary for certain estimates. In general, MATSim expects similar types of variables (e.g., all distances) to be in the same unit wherever they are used. In the following short overview, the most important (expected) units are listed.

Distance Distance units are for example used in links' length. They should be specified in the same unit that the coordinate system uses, allowing MATSim to calculate beeline distances. As the much used UTM (Universal Transverse Mercator) projected coordinate systems (see Section 2.3.3.3) use meters as the unit of distance, this is the most commonly used distance unit in MATSim.

Time MATSim supports an hour:minute:second notation in several places, but internally, it uses seconds as the default time unit. This implies, for example, that link speeds must be specified in distance per second, typically meters per second. One notable exception to this rule are scoring parameters, where MATSim expects values per hour.

Money Money is unit-free. Units are implicitly given by the marginal utility of money (cf. Equation (5.5) below). Thus, when one moves from Germany to Switzerland, the parameter β_m must be changed from "utility per Euro" to "utility per Swiss Franc".

2.3.3.2 Conventions

MATSim uses IDs intensely. These identifiers can be arbitrary strings, with the following exceptions: IDs should not contain any whitespace characters (incl. tabs, new lines, etc.) or commas, semicolons, etc., because those characters are typically used for separating different IDs from each other on ID lists.

2.3.3.3 Coordinate Systems

Preparing Your Data in the Appropriate Coordinate System In several input files, you need to specify coordinates, e.g., for network nodes. For the time being, we strongly advise not to use WGS84 coordinates (i.e., GPS (Global Positioning System) coordinates), or any other spherical coordinates (coordinates ranging from -180 to $+180$ in west-east direction and from -90 to $+90$ in south-north direction). MATSim has to calculate distances between two points in several sections of the code. Calculation of distances between spherical coordinates is very complex and potentially slow. Instead, MATSim uses the simple Pythagoras theorem, but this requires Cartesian coordinate system coordinates. Thus, we emphatically recommend using a Cartesian coordinate system along with MATSim, preferably one where the distance unit corresponds to one meter.

Many countries and regions have custom coordinate systems defined, optimized for local usage. It might be best to ask GIS (Geographic Information System) specialists in your region of interest for the most commonly used coordinate system there and use that for your data.

If you have no information about what coordinate system is used in your region, it might be best to use the UTM coordinate system. This system divides the world into multiple bands, each six degrees wide, and separated into a northern and southern part, which it calls UTM zones. For each zone, an optimized coordinate system is defined. Choose the UTM zone for your region (Wikipedia has a good map showing the zones; an even better alternative is `https://www.geoplaner.com/`) and use its coordinate system.

Telling MATSim About Your Coordinate System For some operations, MATSim must know the coordinate system where your data is located. For example, some analyses may create output to be visualized in Google Earth or by QGIS (Quantum GIS). The coordinate system used while running MATSim can be specified in the config file:

```
<module name="global">
   <param name="coordinateSystem" value="EPSG:32608" />
</module>
```

You have multiple ways to specify the coordinate system you use. The easiest one is to use the so-called "EPSG (European Petroleum Survey Group) codes". Most of the commonly used coordinate systems have been standardized and numbered. The EPSG code identifies a coordinate system and can be directly used by MATSim. To find the correct EPSG code for your coordinate system (e.g., for one of the UTM zones), the website `http://www.spatialreference.org` is useful. Search on this website for your coordinate system, e.g., for "WGS 84 / UTM Zone 8N" (for the northern-hemisphere UTM Zone 8), to find a list of matching coordinate systems along with their EPSG codes (in this case `EPSG:32608`). As an alternative, MATSim can also parse the description of a coordinate system in the WKT (Well-Known Text) format.

Since release 0.8.x, MATSim accepts input data in coordinate systems different from the internal coordinate system. This is achieved by settings of type

```
<module name="network">
   ...
   <param name="inputCRS" value="EPSG:12345" />
```

```
</module>
```

This also works for other input files, e.g., plans files. All coordinates from that file are then, during input, transformed into the coordinate system given in the `global` section of the config.

A third option is to specify the coordinate system directly in the corresponding file. The syntax approximately is

```
...
<network>
   <attributes>
      <attribute name="coordinateReferenceSystem"
         class="java.lang.String">EPSG:31468</attribute>
   </attributes>
   ...
```

The `output_...` files normally contain such an entry, so one can check there. This also works for input files that have a different coordinate system than the `global` coordinate system described above, and it provides the same functionality as the `inputCRS` entry in the config file.

2.3.4 Data Requirements

2.3.4.1 Population and Activity Schedules

As discussed, MATSim needs a file with initial plans as input. All choice dimensions that will not modified later through MATSim's day-to-day learning have to be exogenously estimated here. As of 2021, MATSim has modules for route choice, mode choice, (departure) time choice, and secondary activity location choice, and in consequence everything else needs to be exogenously provided. This concerns, in particular, each person's sequence of activities.

For population generation, two possibilities exist: the comfortable way is to translate a full population census; the more demanding way is to generate a synthetic population (e.g., Guo and Bhat, 2007), based on sample or structure surveys. For MATSim, both methods have been used (e.g., Swiss Federal Statistical Office (BFS), 2000; Müller, 2011).

Travel demand is usually derived from surveys: for Switzerland, from the microcensus (Swiss Federal Statistical Office (BFS), 2006). Newer data sources, such as GPS or smartphone travel diaries, can be used as well (e.g., Zilske and Nagel, 2015).

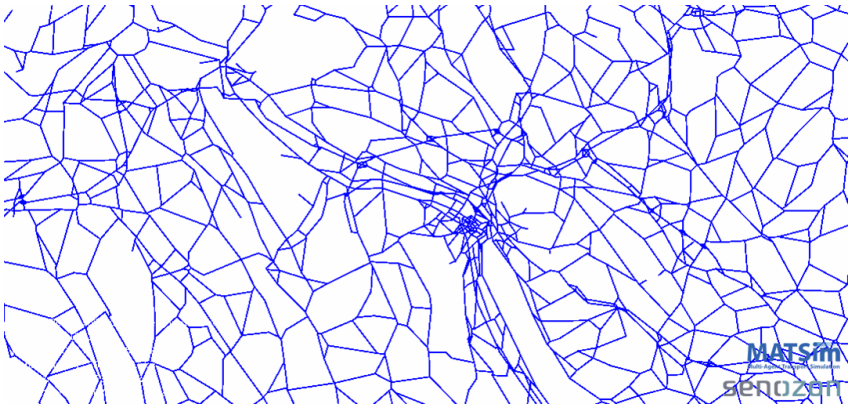
A critical topic in demand and population generation is workplace assignment, as commuting traffic is still a major issue, particularly during peak hours. Switzerland's full census work location was surveyed at municipality level. Such comfortable data bases are rare, however.

Having generated the residential population of the study area, additional demand components might be necessary, for example, cross-border and freight traffic. As these components often cannot be endogenously modeled, MATSim offers the feature to handle different subpopulations differently (Section 4.6). One can specify that border-crossing agents, for example, are not allowed to make destination choices within the study area, or that freight agents are not allowed to change their delivery activity to a leisure activity.

2.3.4.2 Network

In simulation practice, two different network types are used: planning networks and navigation networks (compare Swiss examples in Figure 2.4(a) and Figure 2.4(b) for the Zürich region). The former are leaner and often serve for initial explorative simulation runs, while the latter are often used for policy runs, usually offering far more details, such as bike and even pedestrian links. Data are available from official sources like federal offices, free sources, such as OSM (OpenStreetMap), and commercial sources, including navigation network providers.

(a) Planning network.



(b) Navigation network.

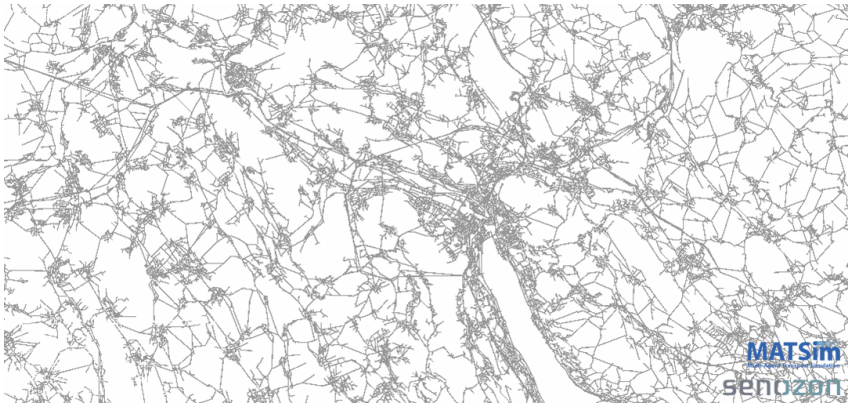


Figure 2.5: Zürich networks.

2.3.5 Open Scenario Input Data

Some example scenarios are under `https://github.com/matsim-org/matsim-code-examples/tree/14.x/scenarios`. More pre-packaged scenarios can be found under `http://matsim.org/open-scenario-data`.

2.4 MATSim Survival Guide

There are many options and possibilities available with MATSim, and finding them can be a daunting exercise. Here are a couple of recommendations, derived from our own frequent use of the system.

1. *Since version 0.9.x, most file paths in the config file are relative to the directory of the config file.*

   This should, in general, make the handling of file paths much easier than in the past. However, older config files may not work any more.

   Somewhat unexpectedly, in most cases URLs can be used as input path names.

2. *Always start and test with a small example.*

3. *Always test large scenarios with one percent runs first (e.g., a randomly drawn subsample of your initial demand).* The MATSim GUI (Figure 2.1) allows creating sample populations with the command `Tools...Create Sample Population`.

   As described in Section 4.4, this requires adaptation of parameters, in particular, the mobsim's `flowCapacityFactor` and `storageCapacityFactor` factors. As shown in Part II, Section 6.3, sample scenarios also require parameter adaption for count data comparisons.

4. *If your set-up does not work any more,* immediately *go back to a working version and proceed from there in small steps.*

5. *Check* `logfileWarningErrors.log`.

6. *Check the comments that are attached to the config file options.*

   One finds them in the file `output_config.xml.gz`, or near the beginning of `logfile.log`.

7. *Try setting as few config file options as possible.*

   This has two advantages: (i) Except for the deliberately set options, your simulation will move along with changed MATSim defaults, and thus with what the community currently considers the best configuration. (ii) You will not be affected by changes in the config file syntax as long as they are different from your own settings.

   The output directory contains an `output_config_reduced.xml`, which is a starting point for such a "minimal" config files.

8. *Use the javadoc documentation that is available from your IDE.*

9. *Search for the latest tutorial via* `http://matsim.org/docs`.

10. *Initially run more complicated modes as teleportation.*

    You can add them as "real" modes later.

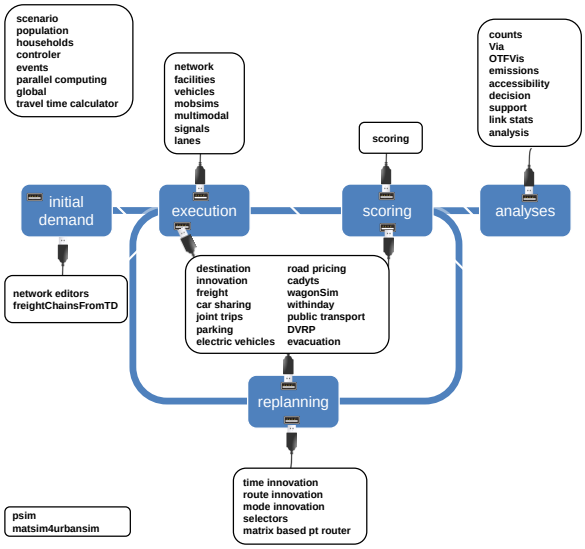
11. *Initially run without extensions.*

    You should add them one by one and verify that they are doing what they should.

3

Available Functionality and How to Use It

Authors: Andreas Horni, Kai Nagel



In this chapter you will learn about possibilities to extend and customize MATSim through *provided* functionality. Chapter 45 describes how you can hook your *own* extensions into MATSim.

3.1 MATSim Modularity

MATSim follows a modular concept, but a "module" is not a very specific term;[1] thus, modules can exist at many levels in a software framework. Also in MATSim, a range of different functionality types, such as sets of configuration options, replanning components, contributions, or even external tools,[2] are sometimes described as modules. It is important to understand the different levels of access stemming from the generally modular architecture.

[1] According to the Merriam-Webster (`http://www.merriam-webster.com`), a module is "one of a set of parts that can be connected or combined to build or complete something" or more specifically "a part of a computer or computer program that does a particular job".

[2] Standalone tools such as the MATSim network editor for JOSM (Java Open Street Map Editor), or the visualizer Via.

3.2 Levels of Access

MATSim currently provides five levels of access, described in the following five sections.

3.2.1 Using MATSim through the GUI with config, network, and population only

To use only the core, one needs to do the following (see Section 2.1):

- Download a MATSim release or a nightly build, by following the respective links at `http://matsim.org/downloads`.
- Obtain a network file and an initial plans file. Small versions can be typed by hand; larger versions should be generated automatically by some computational method.
- Write or edit a config file.[3]
- Click on the MATSim jar file[4] and follow the instructions.

We think that the MATSim core is already quite powerful; for example, synthetic persons already follow full daily plans with a full daily scoring function; thus, opening times for activity types, departure time choice and schedule delay can be investigated.

3.2.2 Using MATSim through the GUI with additional files

Some additional functionality can be used from the GUI by providing additional files, and setting configuration switches accordingly. This concerns most importantly explicit simulation of other vehicles than car (Chapter 11), and schedule-based public transit (Chapter 16).

3.2.3 Writing "Scripts in Java"

If using the MATSim main distribution is not sufficient, the currently recommended way to proceed is to learn how to write MATSim "Scripts in Java". This means to use Java as a scripting language for MATSim; see Section 3.3 for a discussion. The syntax is roughly:

```
... main( ... ) {
    // construct the config object:
    Config config = ConfigUtils.xxx(...) ;
    config.xxx().setYyy(...) ;
    ...

    // load and adapt the scenario object:
    Scenario scenario = ScenarioUtils.loadScenario( config ) ;
    scenario.getXxx().doYyy(...) ; // (*)
    ...

    // load and adapt the controler object:
    Controler controler = new Controler( scenario ) ;
    controler.doZzz(...) ; // (**)
    ...

    // run the iterations:
    controler.run() ;
}
```

[3] Note that between 0.8.x and 0.9.x the convention for *relative* input path names has changed; starting with 0.9.x they start in the directory where the config file resides. Absolute path names are not affected.

[4] This works since the 0.8.x release.

For a working example, see https://github.com/matsim-org/matsim-example-project; this will also contain a valid pom.xml, which will pull in all library dependencies including MATSim itself via Maven. Extension points, especially at (*) and (**), are described in more detail in Chapter 45. Clearly, one needs some Java and IDE experience for this.

Examples for such MATSim scripts-in-java can be found at https://github.com/matsim-org/matsim-code-examples. Note that there are different branches, corresponding to the different MATSim versions.

A useful snippet is

```
Config config;
if ( args==null || args.length==0 || args[0]==null ){
        config = ConfigUtils.loadConfig( "scenarios/equil/config.xml" );
} else {
        config = ConfigUtils.loadConfig( args );
}
config.controler().setOverwriteFileSetting(
        OutputDirectoryHierarchy.OverwriteFileSetting.deleteDirectoryIfExists );
```

3.2.4 Using Contribs or External Extensions

There are MATSim extensions that are not part of the main distribution. They come in two flavours:

- Extensions located in the **contrib** section of the MATSim repository.
- Extensions that are **external** to the MATSim repository.

MATSim extensions can be found via http://matsim.org/extensions.

External extensions tend to be more independent from the MATSim main distribution. This can be a result of more independent tasks—such as input generation or output analysis/visualization—or of more independent API design.

Both for contribs and for external extensions, one needs to read their own documentation for information about their usage. Such documentation can be found via http://matsim.org/extensions.

The MATSim core team recommends again to start from https://github.com/matsim-org/matsim-example-project for using contribs or external extensions, since both types of extensions can be used by adding them into the pom.xml and let Maven do the dependency management.

Clearly, once more this means that one needs to have some Java and IDE experience.

Illustration: otfvis Roughly:

```
... main( ... ) {
    Config config = ConfigUtils.xxx(...) ;
    Scenario scenario = ScenarioUtils.loadScenario( config ) ;
    Controler controler = new Controler( scenario ) ;
    controler.addOverridingModule( new OTFVisLiveModule() ) ;
    // ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
    controler.run() ;
}
```

This should open otfvis, an interactive visualizer attached directly to each mobsim run.

3.2.5 Writing Your Own Extensions

If the existing extensions are not sufficient to plug your own study together, the next option is to write your own extension. Again, we recommend starting from https://github.com/matsim-org/matsim-example-project. Also, one should use the extension points described in Chapter 45, since this is the only way an extension can later become a contribution.

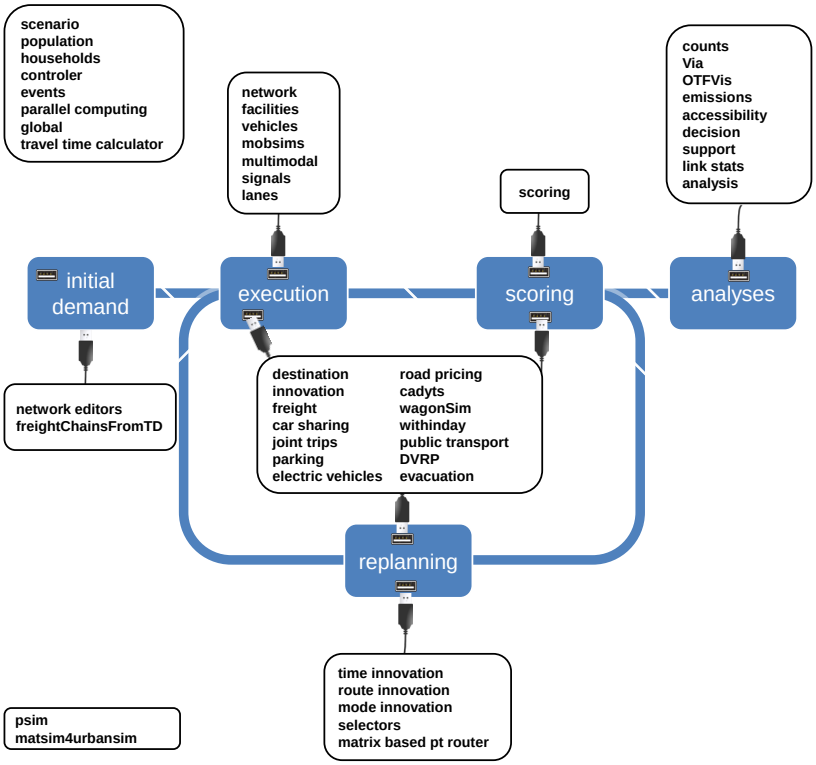


Figure 3.1: MATSim functionality.

3.3 The Ideas Behind this Setup

The setup, as described above, arose from the observation that an-ever growing monolithic MATSim would eventually overwhelm the MATSim team and its core developers group. Therefore, a set-up was sought allowing them to concentrate on central infrastructure, while specific functionality like road pricing, multimodal simulations, signals, additional choice dimensions, or analysis modules could be written and contributed by the community. Clearly, a plug-in architecture had to be the solution, but it took (and still takes) time and effort to make the extension points sufficiently capable and robust.

At the same time, MATSim is a research platform; research investigates innovative questions, which often means that the questions were not foreseen when the code was designed. Quite often, scripting languages are the solution to such problems; for example, python is allowed in QGIS,[5] VISUM (Verkehr In Städten – UMlegung),[6] EMME (Equilibre Multimodal Multimodal Equilibrium), or SUMO (Simulation of Urban Mobility) (via the TraCI interface)[7] for plug-ins. Scala (SCAlable LAnguage) was discussed for MATSim, but ultimately, it was decided to just use Java itself as the scripting language, with the advantage that users between development and MATSim application do not need to learn two languages. In addition, the TU (Technische Universität) Berlin team can continue to teach Java both as an entry point to MATSim and as a general professional skill.

3.4 An Overview of Existing MATSim Functionality

Figure 3.1 shows where common MATSim modules are coupled with the MATSim loop. Some modules have a single connection point (shown around the loop, connected to the respective loop element), while others have multiple connection points (shown in the middle of the circle), and yet others work on a global range (shown on the left upper and lower corners).

Technical details for module usage can be found starting from `http://matsim.org/extensions`.

As a result of the distributed and project- and dissertation-driven MATSim contribution process (see Chapter 44), modules are often implemented for a specific practical purpose, leading to limitations of the respective module. For example, modules might only work for a specific mode, or for a defined calling order. Normally, additional effort is needed to generalize the module; in consequence, the combination of a specific module with other functionality is often not a straight-forward task. This means that a user will have to systematically test any specific combination of modules before productively applying it.

The description of the modules in Chapter 4, and the following chapters, is based on the categorization shown in Table 3.1.

[5] `http://docs.qgis.org/testing/en/docs/pyqgis_developer_cookbook/`
[6] PTV (2011)
[7] `http://sumo.dlr.de/wiki/TraCI`

Table 3.1: MATSim functionality overview.

Global Modules and Global Aspects	Section 4.3
Controler	Section 4.3.1
Events	Section 2.3.2 and 45.5
Parallel Computing	Section 4.3.2
Global	Section 4.3.3
MATSim Data Containers	Section 4.2 and Chapter 6
Network	Section 4.2.1 and 6.1
Population	Section 4.2.2 and 6.2
Counts	Section 6.3
Facilities	Section 6.4
Households	Section 6.5
Vehicles	Section 6.6
Scenario	Section ??
Network Editors	
MATSim JOSM Network Editor	Chapter ??
Map-to-Map Matching Editors in Singapore	Chapter ??
The "Network Editor" Contribution	Chapter ??
Observational Modules	Section 4.7
Travel Time Calculator	Section 4.7.1
Link Stats	Section 4.7.2
Scoring	Section 4.5
Basic Strategy Modules	Section 4.6
Time Innovation	Section 4.6.1.1
Route Innovation	Section 4.6.1.2
Mode Innovation	Section 4.6.1.3
Selectors	Section 4.6.2
Mobsims	
QSim	Section 4.4.1 and Chapter 11
JDEQSim	Section 4.4.2
Individual Car Traffic	
Signals and Lanes	Chapter ??
Parking	Chapter ??
Electric Vehicles	Chapter ??
Roadpricing	Chapter ??
Other Modes Besides Individual Car	
Public Transport	Chapter 16
The "Minibus" Contribution	Chapter ??
Semi-Automatic Tool for Bus Route Map Matching	Chapter ??
Events-Based Public Transport Router	Chapter ??
Matrix-based pt router	Chapter ??
Multi-Modal Contribution	Chapter ??
Car Sharing	Chapter ??
Dynamic Transport Systems	Chapter 23
Commercial Traffic	
Freight Traffic	Chapter ??
wagonSim	Chapter ??
freightChainsFromTravelDiaries	Chapter ??
Additional Choice Dimensions	

Continued on next page

Table 3.1 – *Continued from previous page*

Destination Innovation	Chapter **??**
Joint Trips and Social Networks	Chapter **??**
Socnetgen	Chapter **??**
Within-Day Replanning	
Within-day Replanning	Chapter **??**
Belief Desire Intention (BDI) Framework	Chapter **??**
Automatic Calibration	
Cadyts	Chapter **??**
Visualizers	
Via Visualizer	Chapter **??**
OTFVis Visualizer	Chapter **??**
Analysis	
Accessibility	Chapter **??**
Emissions	Chapter **??**
Interactive Analysis and Decision Support	Chapter **??**
The “analysis” contrib	Chapter **??**
Computational Performance Improvements	
PSim	Chapter **??**
Other Modules	
Evacuation	Chapter **??**
MATSim4UrbanSim	Chapter **??**

4

More About Configuring MATSim

Authors: Andreas Horni, Kai Nagel

```
<module name="global" >
   <param name="numberOfThreads" value="2" />
   ...
</module>
```

4.1 General

This chapter describes configuration options that can be used together with the three basic elements: config, population and network.

MATSim writes configuration files in several locations; for example, in the logfile, in the iteration output directory, or with the `CreateFullConfig` functionality described in Section 2.1.3. As explained in Section 2.4, these files come with comments explaining configuration options. This is often the best source for configuration options.

4.2 MATSim Data Containers

4.2.1 Network

The config file section

```
<module name="network">
   ...
</module>
```

specifies which network file will be used in the simulation (Section 2.1.3 and 2.3.1.1). Further configuration options, e.g., specification of time-variant networks, are presented in Section 6.1.

4.2.2 Population

The config file section

```
<module name="plans">
   ...
</module>
```

specifies which population file with its day plans will be used (Section 2.1.3 and 2.3.1.2). Further configuration options, e.g., specification of arbitrary agent attributes or subpopulations, are presented in Section 6.2.

Further MATSim containers are described in Chapter 6.

4.3 Global Modules and Global Aspects

4.3.1 controler config file section

The controler is an indispensable module for running MATSim; its parameters are set in the config file section

```
<module name="controler" >
    ...
</module>
```

The MATSim run's output directory, its number of iterations and the plans and events output interval can be specified here. The expected mobsim can be defined (Section 4.4). The routing algorithm is defined here by using

```
   <param name="routingAlgorithmType" value="{Dijkstra | FastDijkstra |
     AStarLandmarks | FastAStarLandmarks | SpeedyALT}" />
```

SpeedyALT is was added just before release 13.0 was finalized. In benchmarks it is about a factor of 4 faster FastAStarLandmarks, which was the fastest alternative up to then. It is recommended to use SpeedyALT, but it is not yet the default.

4.3.2 Parallel Computing

MATSim uses multi-threading to accelerate computing speeds. Related configuration parameters can be found in several config modules; they are combined into one section here.

Global Number of Threads The config file section global contains the setting

```
<module name="global" >
   <param name="numberOfThreads" value="2" />
   ...
</module>
```

This number is used in several places; most importantly for innovative strategies, where multiple requests are distributed to multiple threads. A good starting point is using the number of available cores.

Parallel Event Handling The config file section

```
<module name="parallelEventHandling" >
   ...
</module>
```

is used to define the number of threads used for event handling. As described in Waraich et al. (2009), the simulation can be substantially accelerated when using multiple threads for the events handling, which can be a bottleneck in MATSim simulation runs.

Parallel QSim The number of threads for the parallel QSim (cf. Dobler, 2013) can be configured by

```
<module name="qsim" >
   <param name="numberOfThreads" value="10" />
   ...
</module>
```

General Recommendations Generally, computations using threads are not necessarily faster with more threads, which is also true for MATSim. Some experimentation is necessary for each combination of scenario and hardware. Here are some recommendations:

- For the "global" number of threads, a good starting point is the number of available cores.

- It is no longer possible to switch off parallel event handling completely; setting it to '0' or 'null' or '1' achieves the same result. Setting it to values larger than one sometimes leads to performance gains, but they are rarely significant.
- The most sensitive parameter is that for the QSim. For somewhat newer hardware (e.g., Apple Macbook Pro from 2014), using six of the available eight cores for the QSim can make the mobsim more than a factor of two faster and the machine can still be used for office tasks. Experiences with older servers show that one must carefully investigate the number of threads for the mobsim, since using more threads often slows it down (Dobler, 2013).
- Sometimes, High-Performance Computing Clusters are available. Typically, one pays for computation time, either directly, or by a loss of priority, with an amount proportional to the reserved resources, that is, the time the job took to finish, multiplied by the number of reserved cores. In this kind of situation, the number of cores used throughout the whole process should be stable to avoid paying for unused resources. A recommendation in this case is to set the number of threads for the QSim to the best value (see above), say n, parallel events handling to 1, the "global" number of threads to $n + 1$, and submit the job requesting $n + 1$ cores. Also note that fewer threads are almost always better in terms of efficiency. In addition, for both calibration and "what-if" scenario exploration, one typically needs to run a large number of simulations with different parameters or input data. As total RAM memory is usually not an issue on a cluster, it is often more efficient to run a large number of simulations simultaneously with a low number of threads, rather than a low number of simulations with lots of threads.

4.3.3 `global` config file section

In the config file section

```
<module name="global" >
   ...
</module>
```

the simulation's random seed, the "global" number of Java threads (see Section 4.3.2) and the coordinate system (cf. Section 2.3.3) can be defined. Note that no matter if you explicitly define the random seed or not, MATSim always starts from a fixed random seed, which is either the one you define, or an internal constant. That is, if you start the same version of MATSim twice from the same config, you will get the same sequence of random numbers, and thus exactly the same simulation. If you want to change this behavior, you need to change the random seed explicitly.

4.4 Mobility Simulations

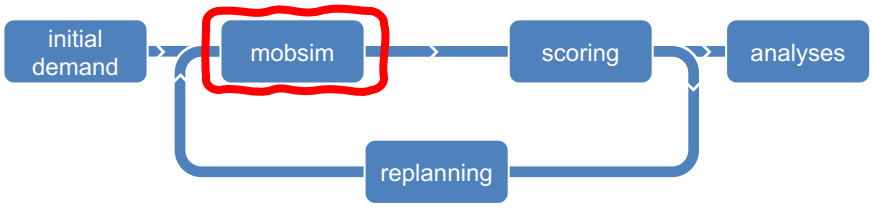


Figure 4.1: Mobsim within MATSim cycle.

An overview of MATSim mobility simulations is given by Dobler and Axhausen (2011). The queue model, which is used both by the QSim and the JDEQSim, implements the following elements:

Storage constraint: vehicles can enter a link only when the link is not full.

Free speed constraint: vehicles can leave a link only after their free speed travel time has passed.

Outflow constraint: vehicles can leave a link only if the outflow capacity has not yet been consumed in the current time step.

See Chapter *Queueing Representation of Kinematic Waves* for details.

4.4.1 QSim

The queue-based and time-step based QSim (Gawron, 1998; Simon et al., 1999; Cetin et al., 2003; Dobler and Axhausen, 2011; Dobler, 2010) is MATSim's default mobsim. Its parameters are set in the config file section

```
<module name="qsim" >
   ...
</module>
```

Important parameters are:

- By specifying

  ```
  <param name="numberOfThreads" value="..."/>
  ```

  QSim can be run in parallel, see Section 4.3.2.

- The parameters

  ```
  <param name="flowCapacityFactor" value="..." />
  <param name="storageCapacityFactor" value="..." />
  ```

  need to be set accordingly when running sample scenarios. For example, for a 10 % sample, these factors need to be 0.1.

- QSim currently supports the following types of traffic dynamics:
  - `queue`: No vehicle can enter when the storage capacity of the link is exhausted.
  - `withHoles`: When a vehicle leaves a link, a so-called hole is created, which travels upstream with 15 km/h, which is a typical kinematic wave speed. A vehicle can enter the link only when a hole is available at its upstream end. The link is initialized with as many holes at its upstream end as it has storage. This models some aspects of kinematic waves.
  - `kinematicWaves`: This goes beyond the `withHoles` setting by also restricting the inflow to a link. The maximum inflow is computed as the maximum flow from a fundamental diagram where both the free flow and the congested branch are assumed as linear.

  Again, see Chapter *Queueing Representation of Kinematic Waves* for details. These settings become increasingly more realistic but less well tested from top to bottom. They are configurable with the parameter

  ```
  <param name="trafficDynamics" value="..." />
  ```

- As shown in Section 11.3, QSim can handle multimodal scenarios.

- The setting

  ```
  <param name="stuckTime" value="..." />
  ```

  determines after how many seconds of non-movement a vehicle is moved across an intersection despite violating the storage constraint of the destination link. This parameter is introduced to resolve grid-locks, i.e., geometrical arrangements where no vehicle can move any more. With the QSim model, it is possible to add vehicles beyond the storage constraint to an overcrowded link. This corresponds to maintaining a minimal flow even under very congested conditions. The default value of this parameter is set to 10, i.e., non-moving vehicles are moved forward after 10 simulation time steps of non-movement. This may seem a rather short time, but systematic investigations (unfortunately never published) have shown that the simulations become, in comparison to traffic counts data, less realistic when this parameter is increased.

4.4.2 JDEQSim

JDEQSim (Waraich et al., 2009) was used for project *KTI Frequencies* (Balmer et al., 2010). It is is a Java reimplementation of DEQSim (Waraich et al., 2009; Charypar et al., 2007b, 2009) and provides parallel event handling, but no parallel simulation (Balmer et al., 2010, p.11). Back-propagating gaps/holes (see above) are supported, but traffic lights, public transport and within-day replanning are not.

To run JDEQSim, the parameter `mobsim` of `controler` config file section must be set to `JDEQSim`, and a config file section

```
<module name="jdeqsim" >
   ...
</module>
```

must be provided.

4.5 Scoring

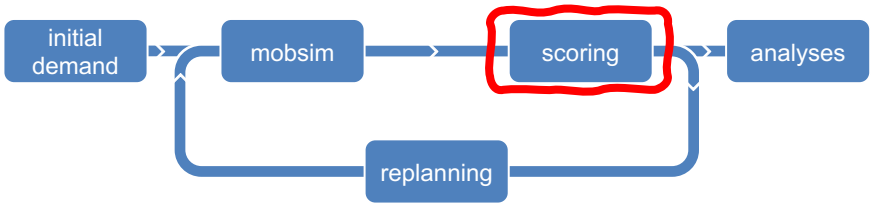


Figure 4.2: Score within MATSim cycle.

The config file section

```
<module name="planCalcScore" >
   ...
</module>
```

specifies the parameters used for scoring agents' plans (Section 2.1.3); parameters are explained in Chapter 5.

4.6 Replanning Strategies

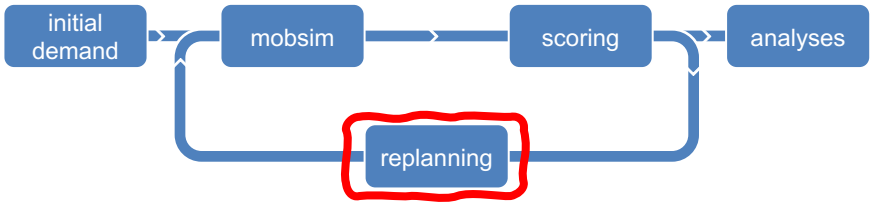


Figure 4.3: Replanning within MATSim cycle.

Replanning strategies are the basic innovation modules available in MATSim. One can differentiate between modules that affect the set of plans that each agent holds (choice set generation), and others that only select between these plans (choice). For a detailed discussion of MATSim in the context of choice modeling, see Chapter *Choice Models in MATSim*.

As already shown in Section 2.2, replanning is defined as shown in the following example:

```
<module name="strategy" >
   <parameterset type="strategysettings" >
      <param name="strategyName" value="ReRoute" />
      <param name="weight" value="0.2" />
   </parameterset>
   <parameterset type="strategysettings" >
      <param name="strategyName" value="TimeAllocationMutator" />
      <param name="weight" value="0.1" />
   </parameterset>
   <parameterset type="strategysettings" >
      <param name="strategyName" value="ChangeSingleTripMode" />
      <param name="weight" value="0.1" />
   </parameterset>
   <parameterset type="strategysettings" >
      <param name="strategyName" value="SelectExpBeta" />
      <param name="weight" value="0.7" />
   </parameterset>
</module>
```

Each module is given a weight, determining the probability by which the course of action represented by the module is taken. The strategies' weights are normalized in case they do not sum to one. In this example, each agent changes her route with weight 0.2, plan timing with probability 0.1, and trip mode also with probability 0.1. Otherwise, the agent chooses a plan from her set of plans according to a logit model. Also see Fig. 4.4.

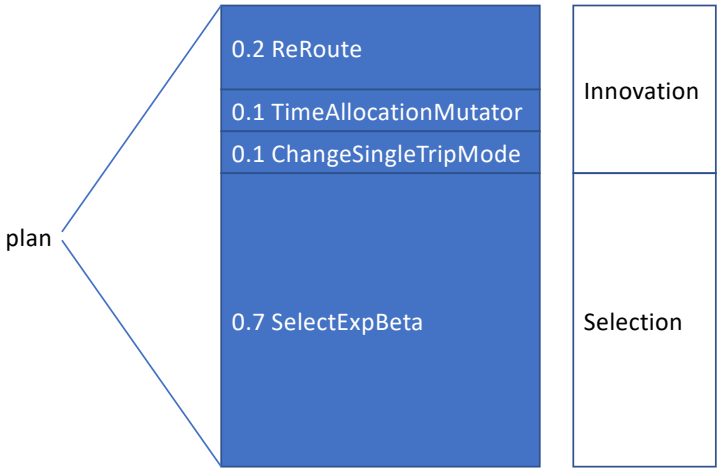


Figure 4.4: Replanning according to the above XML-specification.

In older versions of the config file, you will find a deprecated configuration syntax using numbered strategies.

Please note that combining strategies that are extensions (see Section 3.2), like destination innovation together with public transport, may not always work as expected. Combine them with care, and contact `http://matsim.org/faq` if you are unsure.

4.6.1 Plans Generation and Removal (Choice Set Generation)

This subsection describes config settings that modify the choice set.

4.6.1.1 Time Innovation

Time innovation is applied by adding

```
<param name="strategyName" value="TimeAllocationMutator" />
```

plus its weight as a strategy setting. It is configured by a section

```
<module name="TimeAllocationMutator" >
   ...
</module>
```

This strategy shifts activity end times randomly within a configurable range as described by Balmer et al. (2005); Raney (2005).

4.6.1.2 Route Innovation

Route innovation is applied by adding

```
<param name="strategyName" value="ReRoute" />
```

plus its weight as a strategy setting, and by specifying the routing algorithm in the `controler` config file section (Section 4.3.1). MATSim routing is described by Lefebvre and Balmer (2007).

4.6.1.3 Mode Innovation

Mode innovation is applied by adding

```
<param name="strategyName"
   value="{ChangeTripMode | ChangeSingleTripMode | TripSubtourModeChoice}" />
```

plus its weight as a strategy setting. In the config file, an additional section needs to be added:

```
<module name="{changeMode | changeMode | subtourModeChoice}" >
   <param name="modes" value="car,pt" />
   ...
</module>
```

This would configure mode switches between car and public transit.[1]

`ChangeTripMode` randomly picks one of a person's plans and changes the mode of transport. By default, the supported modes are: driving a car and using public transport. Only one mode of transport per plan is supported. When using different modes for sub-tours on a single day, the `SubtourModeChoice` strategy is required. Optionally, car availability is respected. `ChangeSingleTripMode` randomly picks one of a person's plans and changes one single trip's (picked randomly) mode of transport. In contrast to `ChangeTripMode`, it allows for multiple modes in one plan. By default, supported modes are: driving a car and using public transport. Also, this strategy can (optionally) respect car availability.

Mode innovation is described by Rieser et al. (2009); Meister et al. (2010); Ciari et al. (2008, 2007).

4.6.1.4 Plans Removal

The maximum number of plans per agent is configured by the setting

```
<module name="strategy" >
   <param name="maxAgentPlanMemorySize" value="5" />
   ...
</module>
```

If an agent ends up having more plans, MATSim will start removing plans, one by one, until the maximum number of plans is reached. Plans to be removed are selected by the setting configured by

```
<module name="strategy" >
   <param name="planSelectorForRemoval" value="..." />
   ...
</module>
```

Starting with release 0.8.x, the config file comments give possible options.

This option is not well investigated, cf. Section *Choice Set Generation* in Chapter *Research Avenues*. Per default, the plan with the lowest score is removed if the agent's memory is full.

[1] The additional section was called `changeLegMode` in the past. There should be an error message if you use the wrong name, but please be aware.

4.6.2 Plan Selection (Choice)

Selectors and their weight are also added as strategies by

```
<param name="strategyName" value="KeepLastSelected | BestScore | SelectExpBeta
   ChangeExpBeta | SelectRandom | SelectPathSizeLogit" />
```

Selectors work as follows:

- `KeepLastSelected` keeps the plan selected in the previous iteration.
- `BestScore` selects the plan with the currently highest score.
- `SelectExpBeta` performs MNL (Multinomial Logit Model) selection between plans. It can be configured by the `BrainExpBeta` parameter from the scoring config group[2] being the scale parameter in discrete choice models, as shown in Equation **??**. We recommend keeping this parameter at its default value of 1.0.
- `ChangeExpBeta` changes to a different plan, with probability dependent on $e^{\Delta_{score}}$, where Δ_{score} is the score difference between the two plans. This will also sample from an MNL (see Section *Selection (Choice)* in Chapter *Agent-Based Traffic Assignment*).
- `SelectRandom` performs random selection between the plans.
- `SelectPathSizeLogit` selects an existing plan according to the path size logit model described by Frejinger and Bierlaire (2007). It can be configured by the `PathSizeLogitBeta` parameter from the scoring config group.[3] This selector has never been investigated systematically.

Note that `BestScore` should be used with care; it tends to get stuck with sub-optimal plans: Plans badly scored due to a random fluctuation in one single iteration, e.g., a rare traffic jam, will never be tested again. Thus, we recommend using this only in conjunction with one of the selectors that include a random element.

4.6.3 Innovation Switch-Off

For theoretical (Section *Innovation (Choice Set Generation)* in Chapter *Agent-Based Traffic Assignment*) reasons, it makes sense to eventually switch off the innovative strategies, thus keeping the set of plans for each agent fixed from then on. This behavior can be configured by

```
<param name="fractionOfIterationsToDisableInnovation" value="..." />
```

It makes sense to use this together with MSA (Method of Successive Averages) averaging of the scores (Section 5.3.3).

4.7 Observational Modules

4.7.1 Travel Time Calculator

The routing module, for example, needs travel time estimations for all network links. To keep computational effort feasible, travel time estimations are aggregated into time bins. Parameters of this aggregation, such as bin size, can be specified in the configuration file section `travelTimeCalculator`.

4.7.2 Link Stats

The `linkStats` config file section can specify the output interval of individual links' simulation statistics. It is configurable if the simulated volumes are written per iteration or averaged over multiple iterations.

[2] This is in the scoring config group for historical reasons.
[3] Also in the scoring config group for historical reasons.

As one of their many functions, link stats are used for comparison with count values, as introduced in Section 6.3.

4.8 More Information

The simulation generates a file `output_config.xml`. This file contains all config settings as they were used for the run under consideration, and adds a comment to most config switches. This should be the primary source of information for the meaning of config switches. If some config switch is not explained and you need its interpretation, please ask unter `http://matsim.org/faq` and it will be tried to rectify the situation as quickly as possible.

A complete config dump including all comments is also included in `logfile.log`. Alternatively, complete config files with all switches set to their default values can be generated as explained in Section 2.1.3.

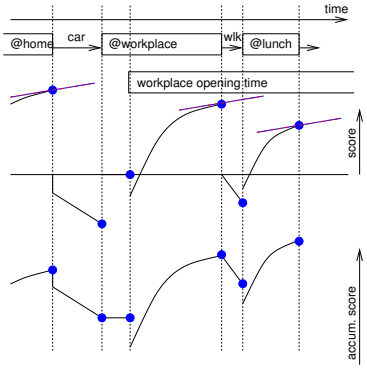
Recall that we explicitly recommend to not use complete config files; rather specify only those switches that you wish to set away from their default settings. The reason is that we change defaults over time to settings that we consider as state-of-the-art according to our research. If you do not allow for this by explicitly setting all switches, then your simulation will over time move away from what we consider the best configuration. Also, you will have compatibility problems with your config file, since we may modify or remove switches over time. The automatically generated `output_config_reduced.xml` can be used as a starting point for a short config file.[4]

[4] Some additional material from that file still has to be removed manually – we are not there yet in fully cleaning this up.

5

A Closer Look at Scoring

Authors: Kai Nagel, Benjamin Kickhöfer, Andreas Horni, David Charypar



5.1 Good Plans and Bad Plans, Score and Utility

As outlined in Section 1.4 and by Figures 1.1 and 1.4, MATSim is based on a co-evolutionary algorithm: Each individual agent learns by maintaining multiple plans, which are scored by executing them in the mobsim, selected according to the score and sometimes modified. In somewhat more detail, the iterative process contains the following elements:

Mobsim The mobility simulation takes one "selected" plan per agent and executes it in a synthetic reality. This may also be called network loading.

Scoring The actual performance of the plan in the synthetic reality is taken to compute each executed plan's score.

Replanning consists of several steps:

1. If an agent has more plans than the maximum number of plans (a configuration parameter), then plans are removed according to a (configurable) plan selector (choice set reduction, plans removal).
2. For some agents, a plan is copied, modified and then selected for the next iteration (choice set extension, innovation).
3. All other agents choose between their plans (choice).

An agent's plans in a given iteration may be considered the agent's **choice set** in that iteration. As a result, steps 1 and 2 of replanning modify the choice set, while step 3 implements the actual **choice** between options. Choice is typically based on the score; higher score plans are more likely to be selected. This is discussed in more detail in Chapters 44.6 and 44.6. For the time being, note that the three steps of replanning must cooperate for the approach to work: the plans removal step should remove "bad" plans, the innovation step should generate "good" plans, and the choice should, in general, select good plans. Here, "good" means "able to obtain a high score in the mobsim/scoring". Fortunately, due to its evolutionary concept, the approach is fairly robust: the innovation step does not always have to generate good solutions; it is sufficient if *some* of the solutions are good and lead to a high score.

With this, it is clear that scoring is a central element of MATSim. Only solutions obtaining a high score will be selected by the agent and survive the plans removal step. Thus, the scoring function needs to be "correct" for a given scenario, meaning, more or less, that plans "performing well" obtain a higher score than plans that "do not perform well". Whether a performance is good or not, is decided, in the end, by travelers living in a region: some may prefer a congested car trip, others may prefer a crowded, but affordable, trip by public transit, while others may prefer using the bicycle, even in bad weather.

The typical way to bridge this gap is to use econometric **utility** functions, for example from random utility models (e.g., Ben-Akiva and Lerman, 1985; Train, 2003) for the score. However, in AI (Artificial Intelligence), utility functions may also be used in a more general way: for example, the score that each individual agent (or the system as a whole) wants to, or should, optimize (Russel and Norvig, 2010). For these reasons, the terms "score" and "utility" are normally interchangeable in the MATSim context. Since we will need the concept of a marginal utility, this chapter will mostly speak of 'utility', since it is a bit unusual to talk about 'marginal score'.

The user can configure numerous parameters to specify the scoring function. When users are ready to extend MATSim in the next part of the book, they will also learn how to plug in their own customized scoring function.

However, because MATSim is based on complete day plans, the application of choice models for parts of day plans only (for example, mode choice) is not straightforward, as detailed in Section **??**. Because of the absence of complete-day utility functions in the literature, MATSim has started with the so-called Charypar-Nagel scoring or utility function (Section 5.2). This scoring function was, at times, modified, extended, or replaced for specific investigations (Section 5.5). Readily applicable estimates for a full-day utility function are not yet available, as discussed in Section **??**.

5.2 The Current Charypar-Nagel Utility Function

5.2.1 Mathematical Form

The first, and still basic, MATSim scoring function was formulated by Charypar and Nagel (2005), loosely based on the *Vickrey* model for road congestion, as described by Vickrey (1969) and Arnott et al. (1993). Originally, this formulation was established for departure time choice. However, all studies performed so far indicate that the MATSim function is also appropriate for modeling further choice dimensions.

5.2.1.1 Basic Function

For the basic function, the utility of a plan S_{plan} is computed as the sum of all activity utilities $S_{act,q}$ plus the sum of all travel (dis)utilities $S_{trav,mode(q)}$:

$$S_{plan} = \sum_{q=0}^{N-1} S_{act,q} + \sum_{q=0}^{N-1} S_{trav,mode(q)} \tag{5.1}$$

with N as the number of activities. Trip q is the trip that follows activity q. For scoring, the last activity is merged with the first activity to produce an equal number of trips and activities.

5.2.1.2 Activities

The utility of an activity q is calculated as follows (see also Charypar and Nagel, 2005, p. 377ff):

$$S_{act,q} = S_{dur,q} + S_{wait,q} + S_{late.ar,q} + S_{early.dp,q} + S_{short.dur,q} \ . \tag{5.2}$$

The individual contributions are defined as follows:

Performing an activity The expression

$$S_{dur,q} = \beta_{perf} \cdot t_{typ,q} \cdot \ln(t_{dur,q}/t_{0,q}) \tag{5.3}$$

is the utility of performing activity q. $t_{dur,q}$ is the performed activity duration, β_{perf} is related to the marginal utility of activity duration (or marginal utility of time as a resource, the same for all activities; see Section 5.2.4), and $t_{0,q}$ is the duration where utility starts to be positive. Opening times of activity locations are taken into account.

Properties of this term at its default setting `typicalDurationScoreComputation=relative` are as follows: With this setting, $t_{0,q}$ is defined by the expression $t_{0,q} := t_{typ,q} \cdot \exp(-1/prio_q)$, where $prio_q$ is a configurable parameter. This allows to re-write Equation (5.3) as

$$= \beta_{perf} \cdot t_{typ,q} \cdot \ln(t_{dur,q}/t_{typ,q}) + \beta_{perf} \cdot t_{typ,q} \cdot \frac{1}{prio_q} \ . \tag{5.4}$$

At the typical duration (i.e. $t_{dur,q} = t_{typ,q}$), the first term is zero, and thus all activity types with the same value of $prio_q$ generate the same relative score of $\beta_{perf}/prio_q$ *per hour* at their typical durations.

In addition, the *slope* of all activities is

$$\frac{\partial}{\partial t_{dur,q}} \left(\beta_{perf} \cdot t_{typ,q} \cdot \ln(t_{dur,q}/t_{typ,q}) + \beta_{perf} \cdot t_{typ,q} \cdot \frac{1}{prio_q} \right) = \frac{\beta_{perf} \cdot t_{typ,q}}{t_{dur,q}}$$

in general, and β_{perf} at their typical durations.

Fig. 5.1 shows how this functional form reacts to changes of t_{typ}, $prio_q$, or β_{perf}. An alternative setting, now deprecated, is discussed in Sec. 5.6.

Waiting The expression

$$S_{wait,q} = \beta_{wait} \cdot t_{wait,q}$$

denotes waiting time spent, for example, in front of a still-closed store; β_{wait} is the so-called *direct* (see Section 5.2.4) marginal utility of time spent waiting; and $t_{wait,q}$ is the waiting time. We recommend leaving β_{wait} at zero; also see Section 5.2.5.

Late arrival The expression

$$S_{late.ar,q} = \begin{cases} \beta_{late.ar} \cdot (t_{start,q} - t_{latest.ar,q}) & \text{if } t_{start,q} > t_{latest.ar,q} \\ 0 & \text{else} \end{cases}$$

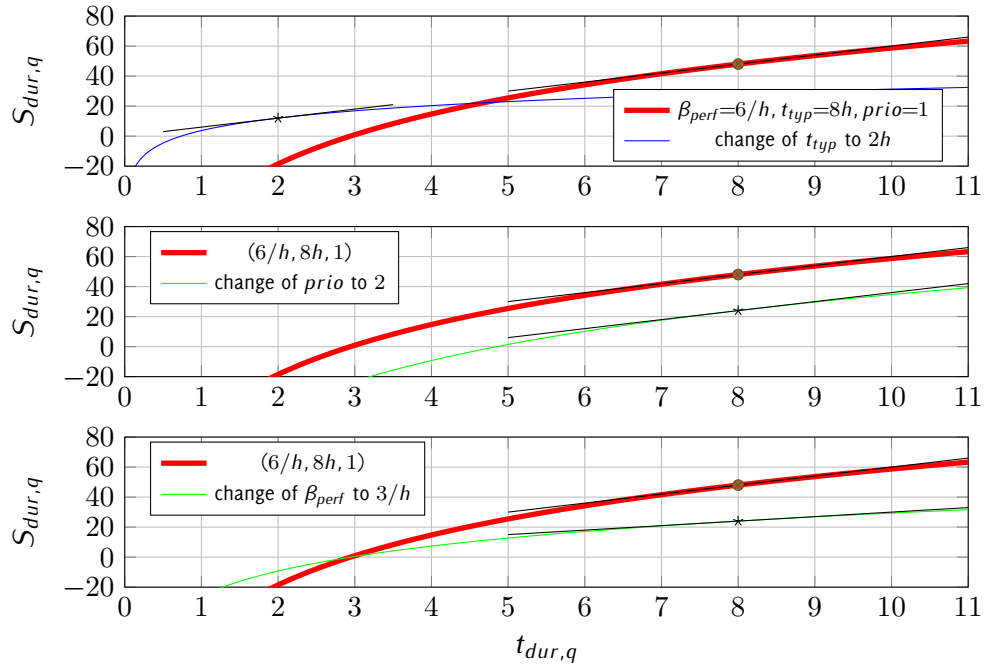


Figure 5.1: Reward for performing an activity when `typicalDurationScoreComputation` is set to `relative`. TOP: Effect of changing the typical duration. MIDDLE: Effect of changing *prio*. BOTTOM: Effect of changing β_{perf}. Note how, in the TOP plot, the score at the typical duration is proportional to the typical duration.

specifies the late arrival penalty, where $t_{start,q}$ is the starting time of activity q, and $t_{latest.ar}$ is the latest possible penalty-free activity starting time—for example, the starting time of the office core hours, or the starting time of an opera or theater performance.

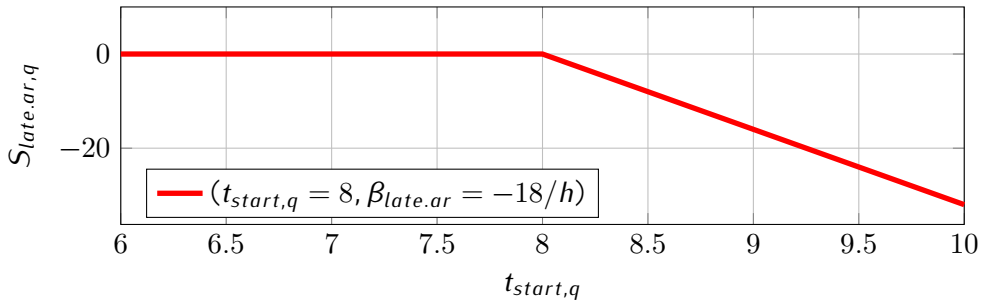


Figure 5.2: Penalty for arriving late.

Early departure The expression

$$S_{early.dp} = \begin{cases} \beta_{early.dp} \cdot (t_{earliest.dp,q} - t_{end,q}) & \text{if } t_{end,q} < t_{earliest.dp,q} \\ 0 & \text{else} \end{cases}$$

defines the penalty for not staying long enough, where $t_{end,q}$ is the actual activity ending time, and $t_{earliest.dp,q}$ is the earliest possible activity end time q. We normally recommend leaving $\beta_{early.dp}$ at zero, except if really good data about this effect is available.

Activity duration too short The expression

$$S_{short.dur,q} = \begin{cases} \beta_{short.dur} \cdot (t_{short.dur,q} - t_{dur,q}) & \text{if } t_{dur,q} < t_{short.dur,q} \\ 0 & \text{else} \end{cases}$$

is the penalty for a 'too short' activity, where $t_{short.dur}$ is the shortest possible activity duration. We normally recommend leaving $\beta_{short.dur}$ at zero, except if really good data about this effect is available.

Config syntax The config syntax (config version v2) is approximately

```
<module name="planCalcScore" >
   <parameterset type="scoringParameters" />
      <param name="subpopulation" value="null" />
      <param name="performing" value="6.0" />
      <param name="waiting" value="-0.0" />
      <param name="lateArrival" value="-18.0" />
      <param name="earlyDeparture" value="-0.0" />
      ...
      <parameterset type="activityParams" >
         <param name="activityType" value="work" />
         <param name="typicalDuration" value="08:00:00" />
         <param name="openingTime" value="07:00:00" />
         <param name="latestStartTime" value="09:00:00" />
         <param name="closingTime" value="19:00:00" />
         ...
      </parameterset>
      ...
   </parameterset>
   ...
</module>
```

There can be different sets of `scoringParameters` for different `subpopulations`.

5.2.1.3 Travel

Travel disutility for a leg q is given as

$$\begin{aligned} S_{trav,q} \quad = \quad & C_{mode(q)} + \beta_{trav,mode(q)} \cdot t_{trav,q} + \beta_m \cdot \Delta m_q \\ & + (\beta_{d,mode(q)} + \beta_m \cdot \gamma_{d,mode(q)}) \cdot d_{trav,q} + \beta_{transfer} \cdot x_{transfer,q} \end{aligned} \tag{5.5}$$

where:

- $C_{mode(q)}$ is a mode-specific constant.
- $\beta_{trav,mode(q)}$ is the *direct* (see Section 5.2.4) marginal utility of time spent traveling by mode. Since MATSim uses and scores 24-hour episodes, this is in addition to the marginal utility of time as a resource (again, see Section 5.2.4).
- $t_{trav,q}$ is the travel time between activity locations q and $q+1$.
- β_m is the marginal utility of money (normally positive).
- Δm_q is the change in monetary budget caused by fares, or tolls, for the complete leg (normally negative or zero).
- $\beta_{d,mode(q)}$ is the marginal utility of distance (normally negative or zero).
- $\gamma_{d,mode(q)}$ is the mode-specific monetary distance rate (normally negative or zero).
- $d_{trav,q}$ is the distance traveled between activity locations q and $q+1$.
- $\beta_{transfer}$ are public transport transfer penalties (normally negative).

- $x_{transfer,q}$ is a 0/1 variable signaling whether a transfer occurred between the previous and current leg.

In addition, there are *daily* monetary and utility constants that are applied if the mode is used at least once. These can be used, e.g., to model car ownership fixed costs.

The config syntax (config version v2) is approximately

```
<module name="planCalcScore" >
   <parameterset type="scoringParameters"/>
      <param name="subpopulation" value="null" />
      ... // parameters that refer to activity scoring
      <param name="marginalUtilityOfMoney" value="1.0" />
      <param name="utilityOfLineSwitch" value="-1.0" />
      <parameterset type="modeParams" >
         <param name="mode" value="car" />
         <param name="dailyMonetaryConstant" value="-5.3" />
         <param name="dailyUtilityConstant" value="0.0" />
         <param name="constant" value="0.0" />
         <param name="marginalUtilityOfDistance_util_m" value="0.0" />
         <param name="marginalUtilityOfTraveling_util_hr" value="-6.0" />
         <param name="monetaryDistanceRate" value="-0.0002" />
      </parameterset>
   ...
</module>
```

Equation (5.5) is the direct utility contribution of travel; see Section 5.2.4 for the the full indirect utility as well as the relation to the VTTS (Value of Travel Time Savings), and Chapter **??** for a more general discussion.

Distance contributes to disutility in two ways. First, it is included in a direct manner via $\beta_{d,mode(q)}$, which is normal for modes involving physical effort, like walking or cycling. Second, distance is also included monetarily via $\beta_m \cdot \gamma_{d,mode(q)}$, which is normal for car or pt mode, where monetary costs increase depending on distance.

Also note that the distance cost rate, $\gamma_{d,mode(q)}$, is given in "monetary units per meters". In Germany, the distance-based marginal costs are approx. $18ct/km$, resulting in `monetaryDistanceRate=-0.00018`, where the unit is Eu/m.

5.2.2 Illustration

Figure 5.3 illustrates the scoring function. Time runs from left to right. The example shows part of an executed schedule, with home, work, and lunch activities, connected by a car and walk leg.

Activities are scored with concave functions, modeling decreasing returns to spending more time at the same activity. Travel, in contrast, is modeled with downward sloping straight lines, where the slope may differ for different modes of transport and there may be an initial offset (alternative-specific constant). Note the delay between arrival at the workplace and workplace opening time, reflected in no score accumulation during that period. Agents accumulate those scores over a day, reflected in the bottom graph.

When one assumes that all other things (particularly travel times) remain fixed and there are no additional constraints, then agents maximize their score when activity durations are such that all activities have the same slope (= the same marginal utility; red lines). This follows from basic economic theory (cf. Section **??**), but can also be seen intuitively; if red lines did not all have the same slope, the agent could gain by extending those activities with steeper slope at the expense of others. Again, clearly this holds only when all other things, in particular travel times, remain constant, and when there are no constraints as, e.g., given by opening times.

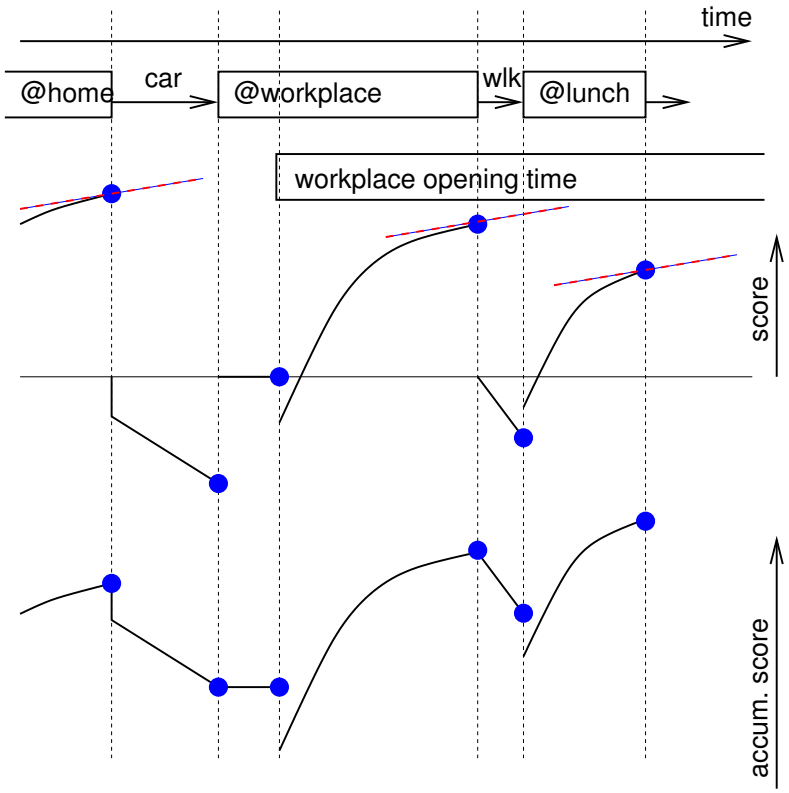


Figure 5.3: Illustration of the scoring function. TOP: Individual contributions of activities and legs. BOTTOM: Score accumulation over a day.

5.2.3 The "Wrapping Around" of the Utility Function

The MATSim mobsim typically starts at midnight and runs until all plans have reached their final activity. By itself, the MATSim approach is not limited to a day. However, as already stated in Section 5.2.1, the standard scoring function assumes that plans "wrap around" to 24-hour days. Thus, the last activity is merged with the first into one activity. For example, if the first activity ends at 7 am and the last activity starts at 11 pm, then it is assumed that this is the *same* activity, with a duration of eight hours.

Note that scoring the two activities separately would lead to a different result, because of the nonlinear (logarithmic) form of the utility of performing. For example, $\ln(1) + \ln(7) = \ln(7) \neq \ln(1+7) = \ln(8)$.

5.2.4 MATSim Scoring, Opportunity Cost of Time, and the VTTS

As a result of the wrap-around concept, travel receives, beyond the typically negative direct marginal utility $\beta_{trav,mode}$, an additional implicit penalty from the **marginal utility of time as a resource**: If travel time could be reduced by Δt_{trav}, the person would not only gain from avoiding $\beta_{trav} \cdot \Delta t_{trav}$, but also from additional time for activities (effect of the opportunity cost of time). The **(total) marginal utility of travel time savings** is thus[1]

$$mUTTS = -\frac{\partial}{\partial t_{trav}} S_{trav} + \frac{\partial}{\partial t_{dur}} S_{dur} = -\beta_{trav} + \beta_{perf} \cdot \frac{t_{typ,q}}{t_{dur,q}} \,, \tag{5.6}$$

where β_{trav} typically is negative, and thus $-\beta_{trav}$ is positive.

[1] The two different signs come from the fact that, say, an decrease in travel times comes along with an increase in activity duration. This could be written formally, but would complicate notation.

At the typical duration of an activity, this is

$$mUTTS\Big|_{t_{dur,q}=t_{typ,q}} = -\beta_{trav} + \beta_{perf} \ ,$$

where it can be imagined q is the activity immediately following the trip (cf. Section **??**). The marginal utility of travel time savings, $mUTTS$, can thus be defined as the indirect effect on the overall time budget, corrected by an offset β_{trav} that denotes how much better, or worse, it is to spend that time traveling, rather than "doing nothing".[2] To differentiate β_{trav} from the indirect effect, it is sometimes called **direct marginal utility** of time spent traveling.

For example, β_{perf} could be $6/h$, and $\beta_{trav,mode1} = -1/h$. This would mean that the base mUTTS would be $6/h$, but spent at mode 1 it would go up to $7/h$. A mode could also be agreeable, e.g. having a $\beta_{trav,mode2} = 1/h$. In this case, mUTTS would be $5/h$.

The marginal utility of travel time savings can be transformed to the more common **VTTS (Value of Travel Time Savings)** by dividing it by the marginal utility of money, β_m:

$$VTTS = \frac{mUTTS}{\beta_m} = \frac{-\beta_{trav} + \beta_{perf} \cdot \frac{t_{typ,q}}{t_{dur,q}}}{\beta_m} \ .$$

At the typical duration of an activity, this is

$$VTTS\Big|_{t_{dur,q}=t_{typ,q}} = \frac{mUTTS}{\beta_m}\Big|_{t_{dur,q}=t_{typ,q}} = \frac{-\beta_{trav} + \beta_{perf}}{\beta_m}$$

This is important for calibration of the utility function.

5.2.5 The Resulting Modeling of Schedule Delay Costs

Arriving Early In the same way as the marginal utility of travel time savings is not only given by $-\beta_{trav}$, but instead by $-\beta_{trav} + \beta_{perf} \cdot \frac{t_{typ,q}}{t_{dur,q}}$, the marginal utility of waiting time savings is given by

$$mUWTS = -\beta_{wait} + \beta_{perf} \cdot \frac{t_{typ,q}}{t_{dur,q}} \ .$$

Even if the direct marginal utility of waiting, β_{wait}, equals zero, then "doing nothing" still eats into the overall time budget and thus incurs the same opportunity cost of time as traveling does. Intuitively, one can imagine that one must leave the previous activity earlier to have a longer waiting time, thus reducing the score of the previous activity.

Thus, as long as one cannot estimate β_{wait} separately from β_{perf}, we recommend leaving β_{wait} at zero.

Arriving Late Arriving late incurs a marginal utility of β_{late}, typically negative. Here, no additional opportunity cost of time is involved. Intuitively, arriving later implies having left the previous activity later. That is: the current activity is shortened by the same amount that the previous activity was extended, leaving the overall score unaffected (cf. Section **??**).

Vickrey Parameters As a result, the Vickrey parameters of α (marginal penalty for arriving early), β (marginal penalty for traveling) and γ (marginal penalty for arriving late) (as defined by Arnott et al.,

[2] This is an approximate statement; in the full theory, the reference marginal utility is not given by "doing nothing", but by a Lagrange multiplier related to the constraint that a day has 24 hours; again, cf. Section **??**.

1990) are consistent with the following equations:

$$\begin{aligned} -\beta_{wait} + \beta_{perf} \cdot \frac{t_{typ,q}}{t_{dur,q}} &= \alpha \\ -\beta_{trav} + \beta_{perf} \cdot \frac{t_{typ,q}}{t_{dur,q}} &= \beta \\ -\beta_{late} &= \gamma \quad . \end{aligned} \tag{5.7}$$

5.3 Implementation Details

This section discusses details of the current implementation of the default MATSim scoring function. The section can be skipped if the reader understands that what has been summarized up to this point is not the full story.

5.3.1 Negative Durations

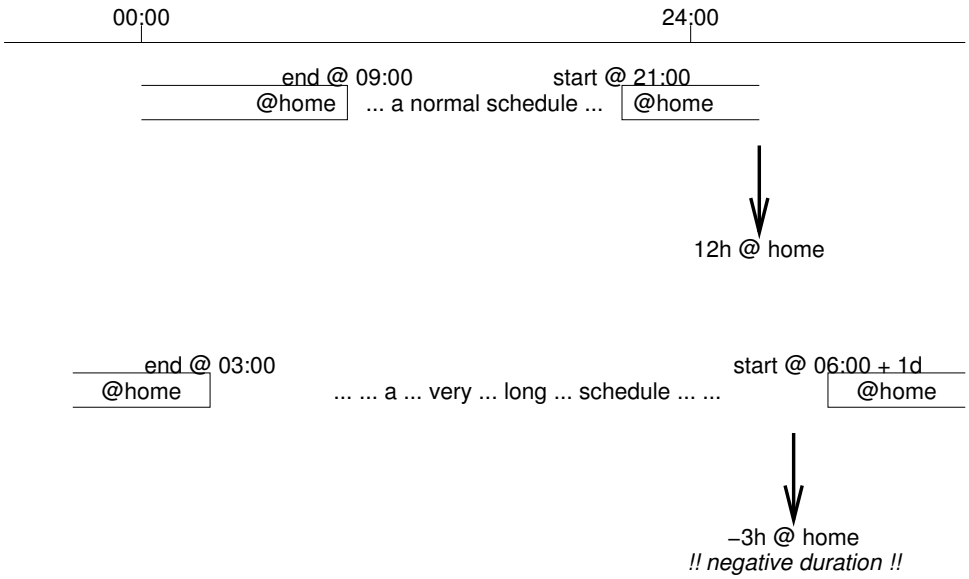


Figure 5.4: Illustration of wrap-around scoring. TOP: Normal situation. BOTTOM: Situation where the final activity of the plan starts at a later time of day then when the first activity ended, resulting in a negative duration of the wrap-around (= overnight) activity..

In MATSim, somewhat oddly, it is possible to have activities with negative durations. This can happen because of the "wrap-around" mechanism, where the last activity of a plan is stitched together with the first activity of the plan, and only that merged activity is scored (cf. Section 5.2.3). In this situation, it can happen that an agent arrives at the last activity of the plan at a later 24-hour-time than when the first activity ended. For example, an agent could stay at home until 3 am (end of first activity), then go through her daily plan including a very late party, and return home at 6 am the next morning (Figure 5.4). In this case, the duration of the wrap-around home activity would be *minus* three hours. The current scoring approach for this situation is to take the slope of the expression $\beta_{perf} \cdot t_{typ,q} \cdot \ln(t_{dur,q}/t_{0,q})$ when it crosses zero, and extend this towards minus infinity (Figure 5.5).[3]

[3]Originally, a score of zero was assigned to these negative duration activities. However, the adaptive agents quickly found out that they could use this to their advantage, expanding this negative duration without a penalty would lead to more time elsewhere, which the agent could use to accumulate score. For an adaptive algorithm, a penalty like this needs to be defined so that it guides the adaptation back into the feasible region. The penalty must increase with increasing negative duration. It

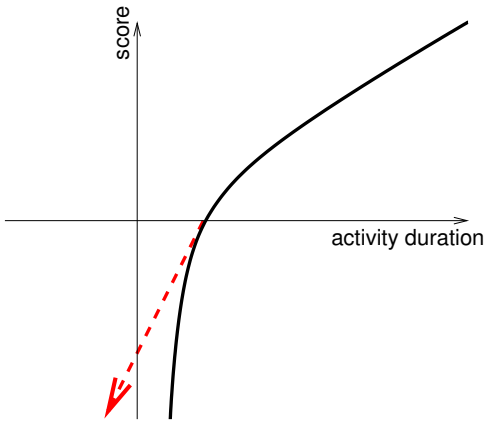


Figure 5.5: Extending the slope when the utility function crosses the zero line to negative durations.

First and Last Activity not the Same Clearly, the wrap-around approach fails if the first and last activity are not the same. The present code does not look at locations, but gives a warning and problematic results if they are of different types.

5.3.2 Score Averaging

The score S that is computed according to the rules given in this chapter is not assigned directly to the plan, rather, it is exponentially smoothed according to

$$S^k = \alpha\, S + (1-\alpha)\, S^{k-1}\,, \tag{5.8}$$

where S^k is the newly memorized score, S^{k-1} is the previously memorized score, S is the score obtained from the plan's execution in the mobsim, and α is a "learning" or "blending" parameter. The default value of α is one; it can be configured by the line

```
<param name="learningRate" value="..." />
```

in the config file.

Non-executed plans just keep their score.

5.3.3 Forcing Scores to Converge

For many situations, both practical and theoretical (see Section **??**), it is desirable that each plan's score converges to its expectation value. Equation (5.8) will not achieve that; it just dampens the fluctuations. A well-known approach to force convergence to the expectation value is MSA:

$$S^m = \frac{1}{m}\, S + \frac{m-1}{m}\, S^{m-1}\,. \tag{5.9}$$

This resembles Equation (5.8), with the important difference that the fixed blending parameter α is now replaced by a variable $1/m$. m is not the iteration number but counts how often a plan was executed and thus scored. This is necessary in MATSim since a plan is not executed and scored in every iteration.

This behavior can be switched on by the following config option:

```
<param name="fractionOfIterationsToStartScoreMSA" value="..." />
```

also needs to be more strongly negative than any score value for a positive activity duration. The latter is, however, impossible to achieve with a logarithmic form, which tends to $-\infty$ as $t_{dur,q}$ approaches zero from above.

This is plausibly used together with innovation switch off (Section 4.6.3), meaning that MSA operates on a fixed set of plans.

5.4 Typical Scoring Function Parameters and their Calibration

The current MATSim default values are

$$\begin{aligned}
\beta_m &= 1 \; utils/monetaryunit \\
\beta_{perf} &= 6 \; utils/h \\
\beta_{trav,mode(q)} &= -6 \; utils/h \\
\beta_{wait} &= 0 \; utils/h \\
\beta_{short.dur} &= 0 \; utils/h \\
\beta_{late.ar} &= -18 \; utils/h \\
\beta_{early.dp} &= 0 \; utils/h.
\end{aligned} \tag{5.10}$$

They are loosely based on the Vickrey bottleneck model (e.g., Arnott et al., 1990).

However, in many situations, and in particular in situations where mode choice in MATSim is enabled, it will be necessary to calibrate the scoring function to your local needs. An additional insight is that, in many of the systems that we model, traveling does not seem to be less convenient than "doing nothing". Thus, the *direct* marginal utility of traveling, β_{trav}, is often close to zero and sometimes even positive (see, e.g., Redmond and Mokhtarian, 2001; Pawlak et al., 2011). Based on this, a possible approach to calibration is as follows:[4]

1. Set $\beta_m \equiv$ `marginalUtilityOfMoney` to whatever is the prefactor of your monetary term in your mode choice logit model.

   If you do not have a mode choice logit model, set to 1.0. (This is the default.)

   This is normally a positive value (since having more money normally increases utility).

2. Set $\beta_{perf} \equiv$ `performing` to whatever the prefactor of car travel time is in your mode choice mode, while changing that parameter's sign from its typical "$-$" to a "$+$".

   If you do not have a mode choice logit model, set to $+6.0$. (This is the default.)

   This is normally a positive value (since performing an activity for more time normally increases utility).

3. Set $\beta_{trav,car} \equiv$ `marginalUtilityOfTraveling...` to 0.0.

It is important to understand this: Even if this value is set to zero, traveling by car will be implicitly punished by the opportunity cost of time: If you are traveling by car, you cannot perform an activity; thus, you are (marginally and approximately) losing β_{perf}. See Section 5.2.4.

4. Set all other marginal utilities of travel time by mode *relative to the car value.*

   For example, if your logit model says something like

   $$\ldots - 6/h \cdot tt_{car} - 7/h \cdot tt_{pt} \ldots,$$

   then

   $$\beta_{perf} = 6 \;, \quad \beta_{trav,car} = 0 \;, \text{ and } \beta_{trav,pt} = -1 \;.$$

   If you do not have a mode choice logit model, set all $\beta_{trav,mode} \equiv$ `marginalUtilityOfTraveling...` values to zero (i.e., same as car).

[4] Different groups have different systems; this one is typical for VSP, using ideas from Michael Balmer.

5. Set distance cost rates `monetaryDistanceRate...` to plausible values, if you have them.

   Note that this needs to be negative: distance consumes money at a certain rate.

6. Use the alternative-specific constants $C_{mode} \equiv$ `constant` to calibrate your modal split.

   (This is, however, not completely simple; one must run iterations and look at the result; especially for modes with small shares, one needs to have innovation switched off early enough near the end of the iterations.)

If you end up having your modal split right, but its distance distribution wrong, you probably need to look at different mode speeds. In our experience, this works better for this than using the $\beta_{trav,mode}$.

Calibrating schedule-based public transport (see Chapter 16) goes beyond what can be provided here.

5.5 Applications and Extensions of the Scoring Function

The default scoring function has been applied and extended for various purposes. Thus, the historical development is accompanied by various conceptual and technical modifications leading to the current utility function described above. This also means that the reported parameter settings in the literature are an indication, not a direct recommendation.

Important applications for large scenarios are described in Chapter **??**.

Special utility functions have been developed for car sharing (see Chapter **??**), social contacts and joint trips (see Chapter **??**), parking (see Chapter **??**), road pricing (see Chapter **??**) and destination innovation (see Chapter **??**), also describing facility loading scoring and inclusion of random error terms.

Future topics, available on an experimental basis, are: a full-blown utility function estimation (Section **??**), inclusion of agent-specific preferences (Section **??**) and application of alternative utility function forms (Section **??**).

5.6 Appendix: Old Version of the Zero Utility Duration

When using the deprecated config setting `typicalDurationScoreComputation=uniform`, $t_{0,q}$ is defined by the expression $t_{0,q} := t_{typ,q} \cdot \exp(-10\,h/t_{typ,q}/prio_q)$, where $prio_q$ is a configurable parameter. This allows to re-write the scoring function from Equation (5.3) as

$$= \beta_{perf} \cdot t_{typ,q} \cdot \ln(t_{dur,q}/t_{typ,q}) + \beta_{perf} \cdot 10\,h \cdot \frac{1}{prio_q} \,.$$

This is evidently such that all activities with the same value of $prio_q$ obtain, at their typical duration, i.e., when $t_{dur,q} = t_{typ,q}$, the same utility value of $\beta_{perf} \cdot 10\,h/prio_q$. The idea was that this would make them equally likely to be dropped in a time shortage situation (Charypar and Nagel, 2005). However, this does not work as intended, since activities receiving this utility value from an activity with a short duration have a larger utility accumulation per time unit than others and are thus dropped later. In consequence, without additional constraints, the "home" activity gets dropped first, which is clearly not plausible. In consequence, the recommendation is to use the `relative` setting. If you insist on using the `uniform` setting, the recommendations are:

- Do not set the `priority` value in the config away from its default value.
- Recognize that the current MATSim default scoring/utility function is not suitable for activity dropping.

With the `relative` setting instead, these deficiencies should be removed. This was, however, never investigated systematically. Fig. 5.6 illustrates the `uniform` setting. Section **??** discusses further alternatives.

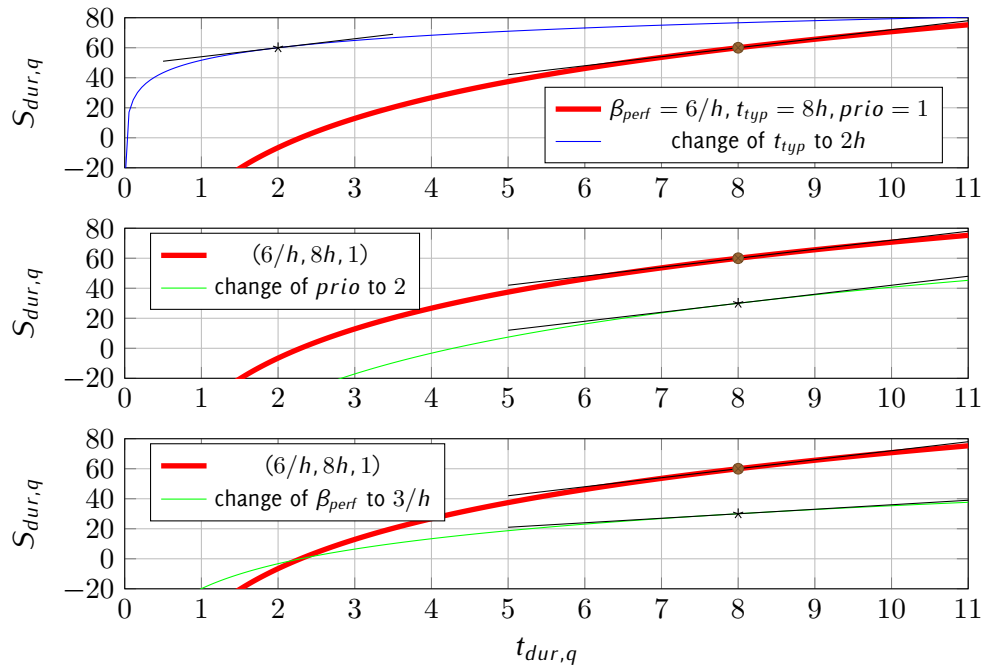


Figure 5.6: Reward for performing an activity when `typicalDurationScoreComputation` is set to `uniform`. TOP: Effect of changing the typical duration. MIDDLE: Effect of changing $prio$. BOTTOM: Effect of changing β_{perf} Note how in the top plot the score at the typical duration just moves to the left, but not down, other than in Fig. 5.1.

Part II
Extending MATSim

6

Additional MATSim Data Containers

Authors: Kai Nagel, Marcel Rieser, Andreas Horni

6.1 Time-Dependent Network

The network container was already described in Section 4.2.1. An important additional feature of the network module is using time-dependent network attributes. Network state changes can thus be considered, as e.g., implied by accidents, or adaptive traffic control, with varying speed limits or driving directions of lanes on multi-lane roads with heavily unbalanced loads over the course of a day. Attributes that can be adapted are "free speed", "number of lanes" and "flow capacity".

The adaptation can be specified by adding the following two lines to the `network` config file section:

```
<param name="timeVariantNetwork" value="true" />
<param name="inputChangeEventsFile" value="path_to_change_events_file" />
```

An example snippet setting the free speed of three network links to zero looks something like this:

```
<networkChangeEvent startTime="03:06:00">
   <link refId="12487"/>
   <link refId="12489"/>
   <link refId="12491"/>
   <freespeed type="absolute" value="0.0"/>
</networkChangeEvent>
```

For a working example, see the file networkChangeEvents.xml in matsim-code-examples.

Alternatively, network change events can be added directly to the code. An example can be found in the RunTimeDependentNetworkExample in matsim-code-examples class.

Note that change values of type `absolute` need to be given in SI units, which means speeds in meters per second and flow capacities in vehicles per second.

6.2 Person Attributes and Subpopulations

The following is valid since release 12.x. Until release 11.x, the subpopulation attribute was stored in a separate file. The approach here is, in our opinion, much more natural.

The population container was also already discussed earlier, in Section 4.2.2. A powerful extension of the standard population can be achieved by making each person a member of a specific subpopulation. This can be achieved by the following syntax:

```
<person id="reroute_0">
```

```
   <attributes>
      <attribute name="subpopulation" class="java.lang.String" >urbanTravelers</attribute>
   </attributes>
   <plan selected="yes">
      ...
```

It is then possible to define different replanning strategies (Section 4.6) for each sub-population, e.g.,

```
<module name="strategy" >
   <parameterset type="strategysettings" >
      <param name="strategyName" value="ChangeTripMode" />
      <param name="weight" value="0.1" />
      <param name="subpopulation" value="urbanTravelers"/>
   </parameterset>
</module>
```

The same is possible with scoring (Section 5), e.g.,

```
<module name="planCalcScore" >
   <parameterset type="scoringParameters" >
      ...
      <param name="subpopulation" value="urbanTravelers" />
      <parameterset type="activityParams" >
         ...
      </parameterset>
      ...
      <parameterset type="modeParams" >
         ...
      </parameterset>
      ...
</module>
```

See RunSubpopulationsExample in matsim-code-examples for an example.

6.3 Counts

By providing a counts input file and configuring the `counts` config file section, MATSim plots link volume comparisons between hourly simulated and counted values for motorized individual traffic (Horni and Grether, 2007).

Simulating sample populations requires scaling simulated volumes by the `countsScaleFactor` parameter, e.g., for a 10 % population this parameter needs to be set to 10.

Input The following listing shows an example of a `counts.xml` input file required for traffic count comparisons.

```
<?xml version="1.0" encoding="UTF-8"?>
<counts name="example" desc="example counting stations" year="2015">
   <count loc_id="2" cs_id="005">
      <volume h="1" val="10.0"></volume>
      <volume h="2" val="1.0"></volume>
      <volume h="3" val="2.0"></volume>
      <volume h="4" val="3.0"></volume>
      <volume h="5" val="4.0"></volume>
      <volume h="6" val="5.0"></volume>
      <volume h="7" val="6.0"></volume>
      <volume h="8" val="7.0"></volume>
      <volume h="9" val="8.0"></volume>
      <volume h="10" val="9.0"></volume>
      <volume h="11" val="10.0"></volume>
      <volume h="12" val="11.0"></volume>
      <volume h="13" val="12.0"></volume>
      <volume h="14" val="13.0"></volume>
      <volume h="15" val="14.0"></volume>
      <volume h="16" val="15.0"></volume>
      <volume h="17" val="16.0"></volume>
      <volume h="18" val="17.0"></volume>
      <volume h="19" val="18.0"></volume>
```

```
      <volume h="20" val="19.0"></volume>
      <volume h="21" val="20.0"></volume>
      <volume h="22" val="21.0"></volume>
      <volume h="23" val="22.0"></volume>
      <volume h="24" val="23.0"></volume>
   </count>
</counts>
```

For a working example, check the `examples/equil` directory in the MATSim directory tree (cf. Section 2.1.1).

It starts with a header containing general descriptive information about the counts, including a year to describe how current the data is. Next, for each link having real world counts data, hourly volumes can be specified. The network-link is referenced by the `loc_id` attribute; in the example, it is link 2. The attribute `cs_id` (counting station identifier) can be used to store an arbitrary description of the counting station. Most often, it is used to note the original real world counting station to simplify future data comparison. The hourly volumes, specified by the hour of the day and its value, are optional: That is, a value does not have to be given for every hour. If, for a counting station, data is only available for certain hours of the day (e.g., only during peak hours), it is possible to omit the other hours from the XML listing. Note that the first hour of the day, from 0:00 am to 1:00 am, is numbered as "1", and *not* by "0" as is often the case in computer science.

Output The counts module prints overview summaries for the whole network, but also analyzes for individual links. Also, a google maps-based visualization is available, showing each station with a its load curve (see the example in Figure 6.1) in a pop-up window.

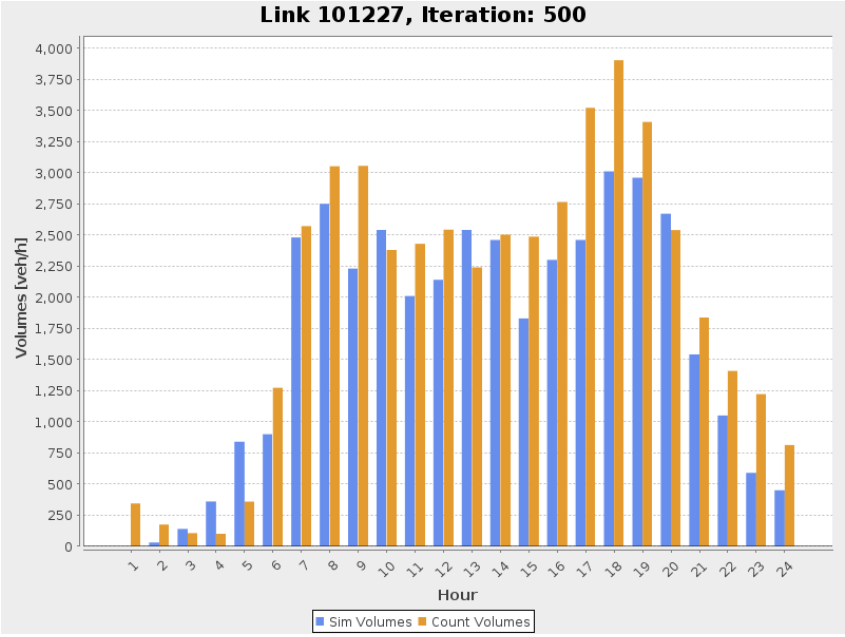


Figure 6.1: Example for a link volumes comparison between simulation and road count values.

Balmer et al. (2009) have performed link volume comparisons for the Zürich scenario, with data based on city level, cantonal level and national level (ASTRA, 2006). Usually, it is helpful to exclude a substantial part of the outer range of the modeled study region in order to remove boundary effects.

6.4 Facilities

Facilities are an optional element of MATSim; some modules, such as the destination innovation module (Chapter **??**), depend on it. If MATSim facilities are used, agents perform their activities in a specific facility attached to a network link.

Facilities are included in the scenario by defining the `facilities` config file section and providing a facilities file, approximately as follows.

```
...
<facilities name="test facilities for triangle network">
   <facility id="1" x="60.0" y="110.0">
      <activity type="home" />
   </facility>
   <facility id="10" x="110.0" y="270.0">
      <activity type="education" />
   </facility>
</facilities>
```

An example is given in `http://matsim.org/javadoc` → main distribution → `RunWithFacilitiesExample` class.

In addition to activities that can be done in the facility, further location attributes, such as opening times, can be specified. A working facilities example file can be found in the MATSim directory tree in the `examples/siouxfalls-2014` directory.

Facilities are mostly used by the MATSim Zürich group, in particular in the Zürich scenario, where they are derived from the Federal Enterprise Census 2001 (Swiss Federal Statistical Office (BFS), 2001) providing hectare level information. Detailed technical description of facilities generation is given by Meister (2008). Comparable data is available in most countries from official sources, such as censuses, and commercial sources, such as navigation network providers, yellow pages publishers or business directories, and last but not least google and OSM (OpenStreetMap, 2015).

Note that loading a facilities file into MATSim by itself does not mean they will be used; the functionality needs to be switched on by other means. Currently, this is only possible by using some class with a `main` method.

6.5 Households

Households are another optional element of MATSim. To load households into a scenario, the config file must contain a section `households`. This section should specify the paths to a file containing households (parameter `inputFile`) and a file containing further household attributes (parameter `inputHouseholdAttributesFile`).[1]

Again, loading the households file does not mean that it is used anywhere in the code; such functionality needs to be switched on separately. Currently, no such functionality can be switched on from the config file alone.

6.6 Vehicles

Vehicles are an optional element of MATSim. To load vehicles into a scenario, a config section

```
<module name="vehicles" >
   <param name="vehiclesFile" value="/path/to/vehicles.xml.gz" />
</module>
```

needs to be added.[2]

[1] There used to be an additional "useHouseholds" config switch. Since release 0.8.x that switch is gone.

[2] There used to be an additional "useVehicles" config switch. Since release 0.8.x that switch is gone.

Once more, just loading the vehicles does not use them; that needs to be configured separately. See Section 11.5 for details.

7

Generation of the Initial MATSim Input

Authors: Kai Nagel, Marcel Rieser, Andreas Horni

As explained in Section 2.3, the minimal MATSim input, besides the configuration, consists of the network and population with initial plans. For illustrative scenarios, all three can be generated with a text editor. For more complicated and/or realistic scenarios, they need to be generated by other methods. People with knowledge in a scripting language may use that scripting language to generate the necessary XML files, possibly honoring the MATSim DTDs (Document Type Descriptions). We ourselves use Java as our scripting language for these purposes. Java is not necessarily the best choice here; this may be discussed elsewhere. We do use it, for the following reasons:

- Most of us also program MATSim extensions and these currently have to be in Java. Thus, using Java as a scripting language for initial input generation saves us the effort of becoming proficient in another programming language.
- The MATSim software, by necessity, already contains all file readers and writers for MATSim input, saving the effort of re-implementing them and one automatically moves forward with file version updates. Additionally, one can directly use the MATSim data containers.
- Once one starts writing MATSim scripts-in-Java (Section 3.2.3), in many situations, it makes sense to modify the input data after reading the files. The programming techniques for this are the same as for other initial input generation.

Part IV will show how initial input was generated on a practical level—discussing, e.g., the different types of original input data—for different scenarios. This section presents MATSim's technical tools for initial input generation.

7.1 Coordinate Transformations in Java

Section 2.3.3.3 has given information about coordinate systems. When programming in Java and MATSim for input data generation, coordinate transformations derived from geotools (Geotools, accessed 2015) can be used. For example,

```
CoordinateTransformation ct =
  TransformationFactory.getCoordinateTransformation("WGS84", "WGS84_UTM33N");
```

would transform data given in WGS84 coordinates to data in UTM coordinates.

7.2 Network Generation

7.2.1 From OpenStreetMap

A fairly standardized way to generate a MATSim network is from OSM data. The process (roughly) goes as follows:

1. Download the necessary `xxx.osm.pbf` file from `http://download.geofabrik.de/osm`.
2. Download a recent Osmosis build from `http://wiki.openstreetmap.org/wiki/Osmosis`.
3. The necessary command to extract the road network (approximately) is:

```
osmosis --read-pbf-fast file=xxx.osm.pbf \
   --bounding-box top=47.701 left=8.346 bottom=47.146 right=9.019 \
   completeWays=true --used-node --write-pbf allroads.osm.pbf
```

   The bounding box can, e.g., be obtained from `http://www.osm.org`; it is in WGS84 coordinates.

4. It makes good sense to add the large roads of a much larger region. The necessary command (approximately) is

```
osmosis --read-pbf-fast file=xxx.osm.pbf --tf accept-ways \
   highway=motorway,motorway_link,trunk,trunk_link,primary,primary_link \
   --used-node --write-pbf bigroads.osm.pbf
```

5. The two files are merged with (approximately) the following command:

```
osmosis --rb file=bigroads.osm.pbf --read-pbf-fast allroads.osm.pbf \
   --merge --write-xml merged-network.osm
```

An example script of how to convert the resulting `merged-network.osm` file into a MATSim network file can be found under `https://github.com/matsim-org/matsim-code-examples/find/11.x` → `RunPNetworkGenerator` class.

7.2.2 From Other Sources

Networks can also be obtained from other sources. An example script of how to convert an EMME network to MATSim can be found under `https://github.com/matsim-org/matsim-code-examples/find/11.x` → `RunNetworkEmme2MatsimExample` class. A problem with EMME network files is that they use user-defined variables in non-standardized ways, meaning that each converter has to be adapted to the specific situation.

Material to read VISUM files can be found by searching for the string "visum" in the code base, but is currently not systematically maintained.

7.3 Initial Demand Generation

7.3.1 Simple Initial Demand

A simple script to generate a population with a single synthetic person with one initial plan can be found under `https://github.com/matsim-org/matsim-code-examples/find/11.x` → `RunPOnePersonPopulationGenerator`. A somewhat larger synthetic population is generated by `RunPPopulationGenerator`.

Note that coordinates in the population need to be consistent with coordinates in the network. Roughly speaking, coordinates mentioned in the population file need to be in the same range as coordinates mentioned in the network. Note that, in the examples presented here, coordinates of the network

generated in Section 7.2.1 are *not* consistent with the demand generated by the RunP*-scripts; these need to be adapted accordingly.

7.3.2 Realistic Initial Demand

A script to illustrate the generation of a more realistic population and initial demand can be found under `https://github.com/matsim-org/matsim-code-examples/find/11.x` → `RunZPopulationGenerator`, generating a sample population from a census file and writing it to a file.

Here, network coordinates generated in Section 7.2.1 *are* consistent with demand generated by the RunZ*-script.

11

QSim and its interaction with routing

Authors: Kai Nagel, Marcel Rieser, Andreas Horni

11.1 Other Modes than Car: Introduction

The MATSim software began with the car mode of transport, since it was then the main mode in many regions. However, the idea of integrating other modes has always been of interest, and in the last years, considerable progress has finally been made.

There are three different types of mode implementations in MATSim:

- teleportation – a simplified mode that approximates travel time and travel distance by other means, and only registers person departures and person arrivals. Cf. Figure 11.2.
- individual routing on the network (= network mode) – something like car, bicycle, or walking, where the decision about turning movements at intersections is made by the persons themselves. Cf. Figure 11.3.
- passenger modes – something like public transport, taxi, or "ride", where the decision about turning movements at intersections is made by someone or something else.

For each mode, these types need to be consistent between routing and QSim – if the router has not computed detailed routing information, then the QSim cannot execute a detailed route. The design is such that a stepwise approach is possible for a new mode:

1. Initially, any new mode can be added by adding it as teleported mode both in routing and in the QSim.
2. As a next step, a detailed router can be used for the mode, but in the QSim it can remain as teleported.
3. Finally, a detailed router can be complemented by a detailed execution in the QSim.

The purpose of this is to be able to include additional modes quickly, and improve their level of detail over time. MATSim in fact runs with teleported modes only.[1]

The following sections describe current MATSim multi-modal capabilities.

[1] Albeit one probably has to include a "car" network so that facilities can be assigned to links, something like "postal addresses" – an approach that was necessary to retrofit many already existing implementations of MATSim.

11.2 Other Modes than Car: Routing Side

First, it is considered how non-car plans, or more specifically non-car routes inside non-car legs, are generated.

11.2.1 Routing Side: Teleportation

Teleportation Routing with Teleported Mode Speed A config entry such as

```
<module name="planscalcroute" >
   <parameterset type="teleportedModeParameters" >
      <param name="mode" value="pedelec" />
      <param name="teleportedModeSpeed" value="4.167" />
      <param name="beelineDistanceFactor" value="1.3" />
      <param name="teleportedModeFreespeedFactor" value="null" />
   </parameterset>
   ...
</module>
```

will generate a teleportation route whose travel distance is 1.3 times the beeline distance, and whose travel time is that distance divided by 4.167 meters per second.

This is useful when teleported mode travel times should not change in tandem with car freespeed travel times, perhaps as a policy change result, or when teleported mode travel times are unrelated to car travel times. One disadvantage: this approach does not take obstacles like water or mountain areas, into account for the teleported modes.

Teleportation Routing with Teleported Mode Free Speed Factor A config entry such as

```
<module name="planscalcroute" >
   <parameterset type="teleportedModeParameters" >
      <param name="mode" value="pt" />
      <param name="teleportedModeFreespeedFactor" value="2.0" />
      <param name="teleportedModeSpeed" value="null" />
      <param name="beelineDistanceFactor" value="null" />
   </parameterset>
   ...
</module>
```

means that if the router encounters a leg with mode pt, it generates a "teleportation" route whose travel distance is the same as, and travel time is twice that of, a freespeed car route.

This models public transit, assuming it travels along roughly the same routes as a car trip, but takes twice as long (cf. Reinhold, 2006).

Teleportation Routing Defaults There are default teleportation routers for the modes *walk, bike, ride* (= in car as passenger) and *pt* (= public transport). These are there for convenience, and so that studies that do not spend a lot of time for their setup get comparable results.

If you define *any* teleportation router yourself, then all default teleportation routers are cleared, and there is a warning message.

To avoid that warning, use `clearDefaultTeleportedModeParams`.

Naming The xml config uses the term "teleported mode (parameters)".

The Java code unfortunately uses the term "mode routing (parameters)" instead.

We will try to change the Java code naming eventually.

11.2.2 Routing Side: Network Modes

The following syntax defines modes for which the router should generate *network* routes, i.e., routes that contain a sequence of links that form a connected path from the origin to the destination:

```
<module name="planscalcroute" >
   <param name="networkModes" value="car, pedelec" />
   ...
</module>
```

The above configuration specifies that plans containing

```
<leg mode="car"...>
```

as well as

```
<leg mode="pedelec"...>
```

will be treated by the network router.[2]

The router will route these modes on links that have the corresponding mode in the network file (Figure 11.1, cf. Section 2.3.1.1).[3]

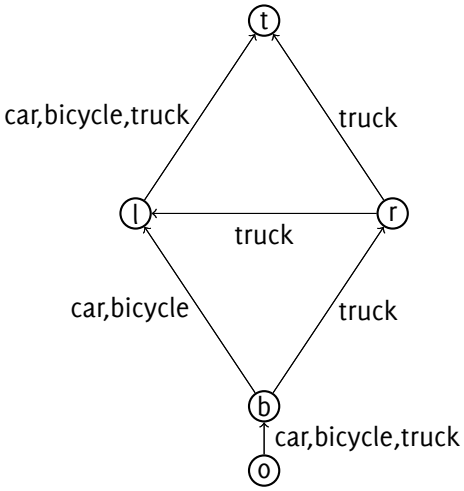


Figure 11.1: Multi-modal network example.

If you try to set a *network* router for a mode for which also a *teleportation* router is defined (see Section 11.2.1), then the code will complain that it cannot define two routers for the same mode. There is a config switch `clearDefaultTeleportedModeParams` that clears all teleportation routers. Evidently, that removes *all* default teleportation routers, so if you still need any of them, you now have to manually enter them.

Often one may want that some vehicle types (e.g. bicycles) have a maximum speed that may be lower than the speed limit of the link. For this, use

```
<module name="qsim">
   <param name="vehiclesSource" value="modeVehicleTypesFromVehiclesData" />
   ...
</module>
```

and then use a mode vehicle type that has a maximum speed (see Section 11.5.1). This option is, for historical reasons, in the `qsim` config group; it does however, also apply to network routing.[4]

[2] Before release 11.0, it used to be the case that one also had to configure `travelTimeCalculator`, setting `separateModes` to true. That is now the default.

[3] Per the network file DTD, "car" is the default mode of each link as long as long as the link's mode field is not explicitly filled.

[4] More precisely, what happens is that that switch makes MATSim generate all mode vehicle types for all persons in the population. And afterwards, both the router and the qsim will use those vehicles.

11.2.3 Routing Side: Passenger Modes

Passenger modes refers to modes that the traveller uses as passenger, e.g. public transit, taxi, or autonomous vehicle. These are discussed in Chapters 16 and 23. They come with their own routers.

11.2.4 Other Routing Options

It is possible to register separate routers for specific modes. This syntax is discussed in Section 45.6.9. The pre-configured extensions and contributions discussed in Section 11.3.4, "multimodal", public transport, taxis, come with corresponding routers.

In addition, the so-called "matrix based pt router" (Chapter **??**) uses a list of transit stops and a matrix of stop-to-stop travel times and travel distances; based on this information, it computes a teleported walk leg to the next stop, another to the destination stop, and a last teleported walk leg to the final destination.

The matrix-based pt router also illustrates that, given the QSim teleportation capability, it is possible to come up with arbitrary routing algorithms for arbitrary modes, as long as they generate (expected) travel times and (expected) travel distances. As said earlier, the teleportation facility of the QSim will just use these two attributes at face value. Although with such an approach neither congestion nor en-route replanning are or can be included, it is flexible and allows a fully modular addition of arbitrary modes without having to interact with the QSim.

11.3 Other Modes than Car: QSim Side

Section 11.2 described several routing options, in particular network routes (consisting of sequences of links) and "teleported" routes (containing only travel time and travel distance). This is now picked up by corresponding functionalities in the QSim.

11.3.1 QSim Side: Teleportation

All modes *not* registered with the QSim in any way will be teleported (Figure 11.2). That is, the QSim will process legs such as

```
<person ...>
   <plan ...>
      ...
      <leg mode="pedelec" >
         <route type="generic" trav_time="00:14:44" distance="2374" ... />
      </leg>
      ...
   </plan>
   ...
</person>
```

For the above plan, the QSim will generate a departure event (for events, see Section 2.3.2) after the end of the previous activity, and an arrival event 14 minutes and 44 seconds later. The leg will be recorded with a distance of 2 374 meters. If distance is not used for scoring (cf. Chapter 5), it can also be left out of the route.

Evidently, this is meant for processing teleported routes as described in Section 11.2.1. It is, however, important to note that this will also process network routes. It is thus possible to introduce a new mode by the following steps:

- First, have the mode teleported both for routing and in the QSim.
- Next, generate network routes during routing, but use teleporting in the QSim. This is, for example, useful for more detailed bicycle routing as long as the mode is not congested.

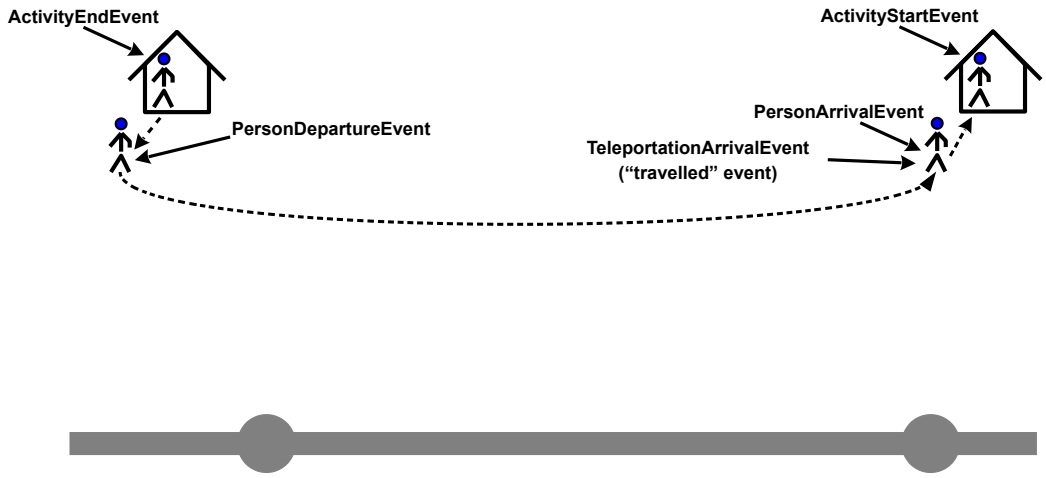


Figure 11.2: Teleported mode.

- Finally, also register the mode as a main mode in the QSim. This is particularly appropriate when the mode is congested.

This makes it possible to introduce a new mode in smaller steps, first as a very approximately handled teleportation mode, and then increasingly more realistic.

Note that the precise coding of a teleported leg has some peculiarities. For example, the start and end link ID need to be given even if they are not really needed. In consequence, it is strongly recommended to use the existing teleportation router, which is parametrized either in the beeline distance, or on the car freespeed travel time. If that is not an option, it is recommended to write your own `RoutingModule`; see `TeleportationRoutingModule` for a starting point.

11.3.2 QSim Side: Network Modes

Multiple modes on the same network are defined by the `qsim` config option of type

```
<module name="qsim">
   <param name="mainMode" value="car,truck,bicycle" />
   ...
</module>
```

This examines the plan leg mode; if that leg mode corresponds to one of the listed main modes, it will generate or obtain a vehicle for that leg and make it enter the network (Figure 11.3).

Evidently, the router needs to have generated network routes as in Section 11.2.2 for these modes.

If mode vehicles are used (Section 11.5.1), and some of them are slower than others, then fast vehicles will not be able to pass slower vehicles. An alternative is to use the so-called `Passing(Vehicle)Q`, which allows faster vehicles to pass slower vehicles while they are not in a queue (Section 11.5.2).

11.3.3 QSim Side: Explicitly Simulated Passenger Modes

With "driver" modes, such as car, bicycle, or also walk, travelers are also drivers, i.e., the entities making decisions about turns at intersections, as well as arrivals (or not) on links. With "passenger" modes, such as public transit or taxi, this changes; the traveler, for example, boards a bus, the bus moves around in the network; the only decision the traveler has to make if she or he wants to get off or not at the current stop. The bus, in turn, is a normal participant in the corresponding traffic system,

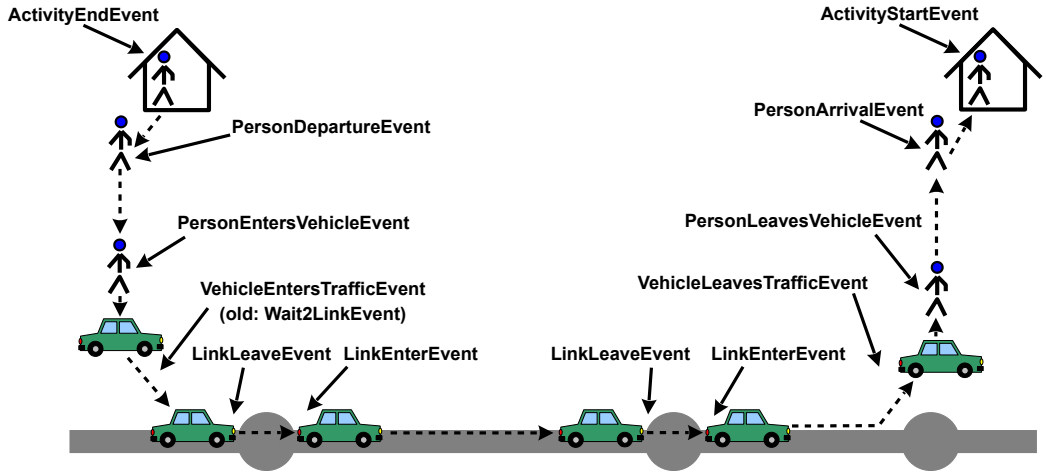


Figure 11.3: Network (= 'main') mode.

i.e., buses and taxis operate on the normal road network and can be caught in the same congestion as cars and trucks.

This is exactly how it works in the MATSim QSim (Figure 11.4); taxis typically operate on the same network as cars; pt vehicles may operate on the same network if their routes are defined so that they use the same links as regular cars; etc. In these cases, their interactions are captured by the simulation.

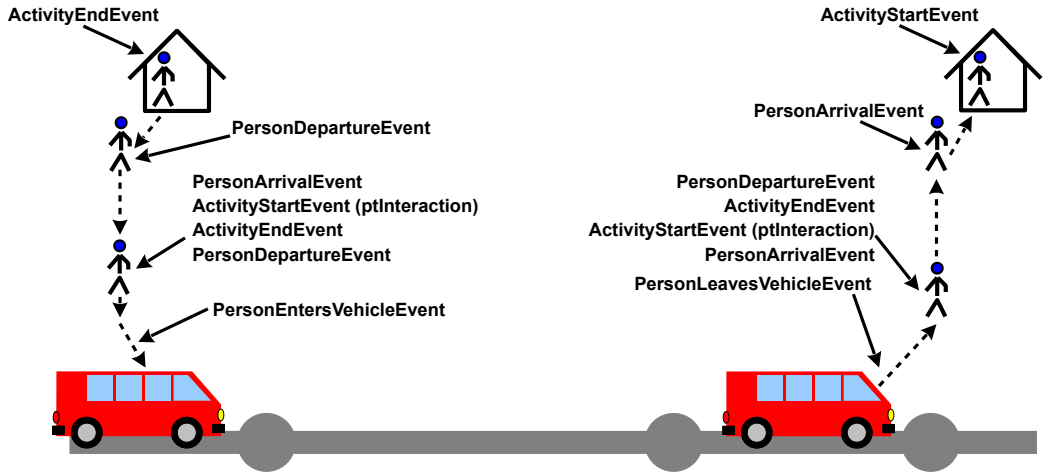


Figure 11.4: Passenger mode.

11.3.4 QSim Side: Departure Handlers

It is possible to register a separate departure handler for each mode; see Section 45.6.10 for the syntax. There are also pre-configured extensions (see `http://matsim.org/extensions`) using this approach:

- The multimodal extension moves all registered modes on separate, congestion-free networks. This is better than teleportation, since the vehicles (or pedestrians) have defined positions at each point in time, meaning that they can also re-plan, e.g., re-route (see Chapter **??**).
- The public transport (pt) extension moves all registered modes with specific public transit vehicles (see Chapter 16).
- The dynamic transport systems (dvrp) extension is able to move a taxicab or av mode as specified in the corresponding section of the config file. See Chapter 23, and the taxicab or av extension.

11.4 Other Modes than Car: Scoring Side

For all modes mentioned in the plans, a corresponding scoring section must exist. See Section 5.2.1 for an example. MATSim will stop with an error message when scoring information for a mode is not available.

11.5 Vehicle Types and Vehicles

11.5.1 Vehicle sources and vehicle IDs

Explicit vehicles

For a variety of reasons—e.g., vehicle-specific emissions calculations (Chapter **??**), or vehicle-specific maximum speeds (see below)—it may become necessary to assign different vehicle types to different persons, modes, or trips. The (arguably) most "honest" approach to vehicles, in terms of micro-simulation, is to generate a synthetic vehicle fleet. Each leg/route would then have to know which vehicle it wants to use. This is indeed possible with the planned vehicle ID that MATSim route objects can store. This functionality is switched on by first loading an additional vehicles file (see Section 6.6) and then configuring the QSim as

```
<module name="qsim">
   <param name="vehiclesSource" value="fromVehiclesData" />
   <param name="usePersonIdForMissingVehicleId" value="false" />
   ...
</module>
```

(available since release 0.8.x). This states that every time the QSim needs a vehicle with a specific ID, it will search for it in the vehicles data container, throwing an exception if it is not found there.

The MATSim design is slowly moving towards explicit vehicles. If "mode vehicles" are switched on (see Section 11.5.1), then a vehicle for each mode is generated for each synthetic person.

These are afterwards also written to file, which means that they are available for possible analysis.

A Fallback for Routes that do not Contain Vehicle IDs

In the above approach, vehicular routes need to provide vehicle IDs, otherwise the QSim will throw an exception. Since algorithms to compute and maintain vehicle IDs during replanning are currently not well developed within MATSim, an alternative is to assume that persons use a vehicle with the same ID as the person. This is switched on by

```
<module name="qsim">
   <param name="usePersonIdForMissingVehicleId" value="true" />
   ...
</module>
```

which is also the current default. With this configuration, it will still search for the vehicle ID in the route, but if this is unavailable, it will instead use the person ID as vehicle ID. Without additional configuration, it will then still search for the vehicle under that ID in the vehicles file.

Alternative Vehicles Source: defaultVehicle

A default vehicles source is defined by

```
<module name="qsim">
   <param name="vehiclesSource" value="defaultVehicle" />
   ...
</module>
```

This generates a default vehicle (typically a medium-sized 4-seater) every time a vehicle is needed and is currently the default configuration.

Alternative Vehicles Source: modeVehicleTypesFromVehiclesData

So-called mode vehicles are configured by

```
<module name="qsim">
   <param name="vehiclesSource" value="modeVehicleTypesFromVehiclesData" />
   ...
</module>
```

For this, the vehicles file (Section 6.6) needs to contain mode specific vehicle *types*, approximately as follows:

```
<vehicleType id="car">
   <maximumVelocity meterPerSecond="16.67"/>
   <passengerCarEquivalents pce="1.0"/>
   ...
</vehicleType>
 <vehicleType id="bicycle">
   <maximumVelocity meterPerSecond="4.17"/>
   <passengerCarEquivalents pce="0.25"/>
   ...
</vehicleType>
```

See config-with-mode-vehicles.xml in matsim-code-examples for a working example.

This approach can also be be programmed as script-in-Java; see RunMobsimWithMultipleModeVehiclesExample in matsim-code-examples. Simulation experiments using this feature have been performed for the Patna scenario as reported in Chapter **??**.

11.5.2 Vehicle Placement and Behavior

Vehicle persistence

Vehicles need to be available where they are needed. In the real world it is, for example, difficult to perform a trip by car, then another (non-circular) trip by public transit and then make another trip with the same car as before, since the car will not be available at that location. The QSim is able to enforce such behavior, with the setting

```
<module name="qsim">
   <param name="vehicleBehavior" value="exception" />
   ...
</module>
```

This means that if a necessary vehicle is not available at the location where it is needed by the traveler, the QSim throws an exception and aborts. This could be useful to debug a simulation where consistent vehicle behavior is desired. Alternatively, synthetic travelers could have explicit within-day replanning strategies (see Chapter **??**) to cope with unexpectedly unavailable vehicles. In both cases, any attempt to use an unavailable vehicle points to an error in the behavioral logic of the simulation.

In many standard situations, the above behavior will be too strict. For example, a vehicle may be shared between family members, but one member will be late in returning a vehicle. For such situations,

```
<module name="qsim">
   <param name="vehicleBehavior" value="wait" />
   ...
</module>
```

may be an option. Here, a driver will wait if a vehicle is not available. Errors in the coordination logic, i.e., very long waits, will be punished via the MATSim scoring logic, thus leading to more robust coordinations.

A final alternative is

```
<module name="qsim">
   <param name="vehicleBehavior" value="teleport" />
   ...
</module>
```

With this setting, vehicles will be teleported to locations where they are needed.

Initial Vehicle Placement

For vehicle behavior of type `exception` and type `wait`, vehicles need to be at the correct location when the QSim starts. Here, the default simulation currently places all vehicles at the home location—for other variants, some additional code needs to be written or used, such as for the car sharing extension (Chapter **??**).

PassingQ

Once various vehicles have different maximum speeds, the standard QSim, even with multiple main modes, is no longer sufficient, since it uses FIFO (First In, First Out) as the queuing discipline, meaning that fast vehicles cannot pass slower vehicles. Here, the so-called `Passing(Vehicle)Q` can be used instead. It replaces the FIFO sorting criterion—where vehicles are sorted by the sequence in which they arrive on the link—by a sorting employing the so-called earliest link exit time, computed from link enter time and freespeed travel time. Now, using the minimum of vehicle and link maximum speeds, the freespeed travel time can be differentiated between vehicles, allowing fast vehicles to obtain an earlier link exit time, even if they enter later than slow vehicles. Details and resulting fundamental diagrams are given by Agarwal et al. (2015).

This option can be enabled by using

```
<module name="qsim">
   <param name="linkDynamics" value="passingQ" />
   ...
</module>
```

in the `qsim` section of the config file.

11.6 Other

The simulation is able to handle time-variant networks (Lämmel et al., 2010, see Section 6.1), within-day replanning (Dobler, 2009, see Chapter **??**) and traffic lights (Neumann, 2008; Grether et al., 2011, 2012, see Chapter **??**). An earlier multimodal approach, targeted at overcoming the teleportation estimates of non-motorized modes, and particularly focused on pedestrians, is presented in Chapter **??**.

16

Modeling Public Transport with MATSim

Author: Marcel Rieser

16.1 Basic Information

Entry point to documentation:
`https://matsim.org/docs` → Tutorials → Simulation of public transport

Invoking the module:
The module is invoked by enabling it in the configuration.

Selected publications:
Rieser (2010)

16.2 Introduction

Public transport—or (public) *transit* as it is sometimes called—plays an important role in many transport planning measures, even those initially targeting only non-transit modes. By making other modes more or less attractive (e.g., by providing higher capacity with additional lanes, allowing higher speeds, or charging money by setting up area road pricing), travelers might reconsider their mode choice and switch to public transport (*pt*) from other modes, or vice versa. Such changes can also occur when transit infrastructure is changed; additional bus lines, changed tram routes with different stops served, or altered headways—all are important for travelers on specific lines, or public transport in general. Around 2007, interest grew in extending MATSim to support detailed simulation of modes other than private car traffic, particularly public transport.

In a first step, MATSim was extended so that modes other than *car* would be teleported; agents would be removed from one location and placed at a later point of time—corresponding to estimated travel time—at their destination location, where they could commence their next activity. Together with a simple mode-choice module, randomly replacing all transport modes in all plan legs and a simple travel time estimation for modes different than *car*, first case studies resulting in modal share changes were performed using MATSim (Rieser et al., 2009; Grether et al., 2009). This teleportation mode is now available, by default, in MATSim and still a very good fallback option to get a multimodal scenario up and running with as little data as possible.

In a second step, QSim was extended to support detailed simulation of public transport vehicles serving stops along fixed routes with a given schedule (Rieser, 2010). The next section describes, in more detail, data required and resulting features for this detailed public transport simulation.

16.3 Data Model and Simulation Features

MATSim supports very detailed modeling of public transport; transit vehicles run along the defined transit line routes, picking up and dropping off passengers at stop locations, while monitoring transit vehicles' capacities and maximum speeds. Data used to simulate public transport in MATSim can be split in three parts:

- stop locations,
- schedule, defining lines, routes and departures, and
- vehicles.

This data is stored in two files; vehicles are defined in one file, stop locations and schedule in another. Examples of such files can be seen in Section 16.4.1 and Section 16.4.2, respectively.

The data model is comparable to other public transport planning software, but simplified in several respects. A line typically has two or more routes: one for each direction, and additional routes when vehicles start (or end) their service at some point on the full route (coming from, or going to, a depot). Each transit route contains a network route, specifying on which network links the transit vehicle drives, as well as a list of departures, providing information about what time a vehicle starts at the first route stop. A route also includes an ordered list of stops served, along with timing information specifying when a vehicle arrives or leaves a stop. This timing information is given as offsets only, to be added to departure time at the first stop. Each departure contains the time when a vehicle starts the route and a reference to the vehicle running this service. Because timing information is part of the route, routes with the same stops sequence may exist, differing only in time offsets. This is often the case with bus lines, that take traffic congestion and longer rush hour waiting times at stops into account in the schedule.

Stop locations are described by their coordinates and an optional name; they must be assigned to exactly one line of the network for the simulation. Thus, they can be best compared to "stop points" in VISUM. There is, currently, no logical grouping of stop locations to build a "stop area"; this is a cluster of stops often sharing the same name, but located on different intersection arms, served by different lines, many with transfer corridors for passengers.

Each vehicle belongs to one vehicle 'type', which describes various characteristics, like seating and standing capacity (number of passengers), its maximum speed and how many passengers can board or depart a vehicle per second.

This data model already supports several advanced public transport modeling aspects: varying travel speeds along routes during different times of day (important for improved simulation realism), using diverse vehicle types on routes at different times of day (interesting for schedule economic analysis) and re-using transit vehicles for multiple headways along one or different routes (allows vehicle deployment planning optimization, or research on delay-propagation effects).

With these data sets, the QSim will simulate all transit vehicle movements. The vehicles will start with their first route stop at the given departure time, allow passengers to enter and then drive along their route, serving stops. At each stop, passengers can enter or leave the vehicle. The simulation generates additional, transit-related events whenever a transit vehicle arrives or departs at a stop, when passengers enter or leave a vehicle, but also when a passenger cannot board a vehicle because its capacity limit is already reached. This allows for detailed analyses of MATSim's public transport simulations.

For passengers to use public transport in MATSim, they must be able to calculate a route using transit services. For this, MATSim includes a public transport router that calculates the best route to the desired destination with minimal cost, given a departure time. Costs are typically defined only as travel time and a small penalty for changing lines, but other, more complex cost functions could be used.

The routing algorithm is based on Dijkstra's shortest path algorithm (Dijkstra, 1959), but modified to take multiple possible transit stops, around the start and end coordinates, into account to find a route. Multiple start and end stops must be considered to generate more realistic transit routes; otherwise, agents could be forced to travel first in the wrong direction, or wait at an infrequently served bus stop, instead of going a bit further to a busy subway stop location. By modifying the shortest path algorithm to work with multiple start and end locations, a considerable performance gain was achieved when compared to the basic (and somewhat naive) implementation that calculated a route for each combination of start/end location and then chose the best outcome.

16.4 File formats

16.4.1 transitVehicles.xml

To simulate public transport in MATSim, two additional input files are necessary. One is `transitVehicles.xml`, which describes vehicles serving the lines: big buses, small buses, trains or light rail vehicles and description of each vehicle's passenger transport capacity.

The PT (Public Transport) vehicle description is split into two parts. In the first part, vehicle types must be described, specifying how many passengers a vehicle can transport. (Note that the term "vehicle" can refer to multiple vehicles in reality, e.g., a train with several wagons should be specified as one long vehicle with many seats). In the second part, all actually used vehicles must be listed. Each vehicle has an identifier and refers to a previously defined vehicle type. The following shows an example of a such a file, describing one vehicle type, and then two vehicles of that type.

```
<vehicleDefinitions ... />
   <vehicleType id="1">
      <description>Small Train</description>
      <capacity>
         <seats persons="50"/>
         <standingRoom persons="30"/>
      </capacity>
      <length meter="50.0"/>
      <passengerCarEquivalents pce="3.0"/>
   </vehicleType>
   <vehicle id="tr_1" type="1"/>
   <vehicle id="tr_2" type="1"/>
</vehicleDefinitions>
```

16.4.2 transitSchedule.xml

The second, rather complex, file necessary to simulate public transport is `transitSchedule.xml`, containing information about stop facilities (bus stops, train stations, or other stop locations), and transit services.

In the first part, stop facilities must be defined; each one is given a coordinate, an identifier and a reference to a network link. The stop can only be served by vehicles driving on that specified link. It is also possible to specify both a name for the stop and whether other vehicles are blocked when a transit vehicle halts at a stop. This last attribute is useful when modeling e.g., different bus stops, where one has a bay, while at another, the bus must stop on the road.

After stop facilities, transit lines, their routes and schedules are described. This is a hierarchical data structure; each line can have one or more routes, each with a route profile, network route, and list of departures. The following listing is an example of a basic, but complete transit schedule.

```
<transitSchedule>
   <transitStops>
      <stopFacility id="1" x="990.0"  y="0.0"   name="Adorf"
           linkRefId="1" isBlocking="false"/>
      <stopFacility id="2" x="1100.0" y="980.0" name="Beweiler"
           linkRefId="2" isBlocking="true"/>
      <stopFacility id="3" x="0.0"    y="10.0"  name="Cestadt"
           linkRefId="3" isBlocking="false"/>
   </transitStops>
   <transitLine id="Blue Line">
      <transitRoute id="1">
         <description>Just a comment.</description>
         <transportMode>bus</transportMode>
         <routeProfile>
            <stop refId="1" departureOffset="00:00:00"/>
            <stop refId="2" arrivalOffset="00:02:30" departureOffset="00:03:00"
                                                     awaitDeparture="true"/>
            <stop refId="3" arrivalOffset="00:05:00" awaitDeparture="true"/>
         </routeProfile>
         <route>
            <link refId="1"/>
            <link refId="2"/>
            <link refId="3"/>
         </route>
         <departures>
            <departure id="1" departureTime="07:00:00" vehicleRefId="12"/>
            <departure id="2" departureTime="07:05:00" vehicleRefId="23"/>
            <departure id="3" departureTime="07:10:00" vehicleRefId="34"/>
         </departures>
      </transitRoute>
   </transitLine>
</transitSchedule>
```

Each transit line must have a unique ID. Each transit route has an ID which must be unique within that one line, allowing the same route ID to be used with different lines. The `transportMode` describes network links where the line runs. (Actually, this is not yet in force, although it might be in the future. It would currently be possible to let a bus run on train links in the simulation.)

The `routeProfile` describes the stops this route serves and the timings at these stops. The `route` itself describes the series of network links the transit vehicle's driver must navigate, often referred to as network route. Note that the complete route, i.e., all links the vehicle traverses, must be listed in the route, not only those with stops. All specified stops should occur along this route in correct order. Time offsets given for each stop in the `routeProfile` describe relative time offsets to a departure time from the first stop, listed unter `departures`. If a bus has a departure at 7 am, and stop 2 has a `departureOffset` of 3 minutes, this means that the bus is expected to depart at 7:03 am from the specific stop. All stops in the route profile must have a departure offset defined, except for the last one. All stops, except the first one, can, optionally, have an arrival offset defined. This is useful for large trains that stop for several minutes at a station; helping the routing algorithm find connecting services at the correct time, namely the expected train arrival time.

As the last part of a transit route description, a `departures` list should be given. Each departure has an ID, which must be unique within the route, giving the departure time at the first stop of the specified route profile. The departure also specifies the vehicle (which must be defined in the previous transit vehicle list) with which the service should be run.

Because of its complexity, transit schedules often contain small mistakes that will result in an error when the simulation runs. Typical examples include: missing links in the network route, or incorrect defined stop order on the network route. To ensure that a schedule avoids such issues before the simulation starts, a special validation routine is available. It is easiest to use from the GUI, see Figure 16.1;

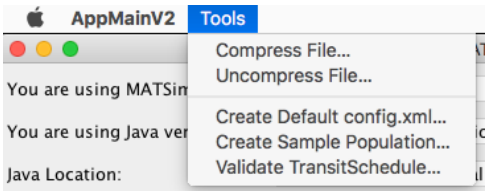


Figure 16.1: MATSim GUI tools.

alternatively, it can be run from the command line.[1] If run, this validator will print out a list of errors or warnings, if any are found, or show a message that the schedule appears to be valid.

16.4.3 Events for a public transit trip

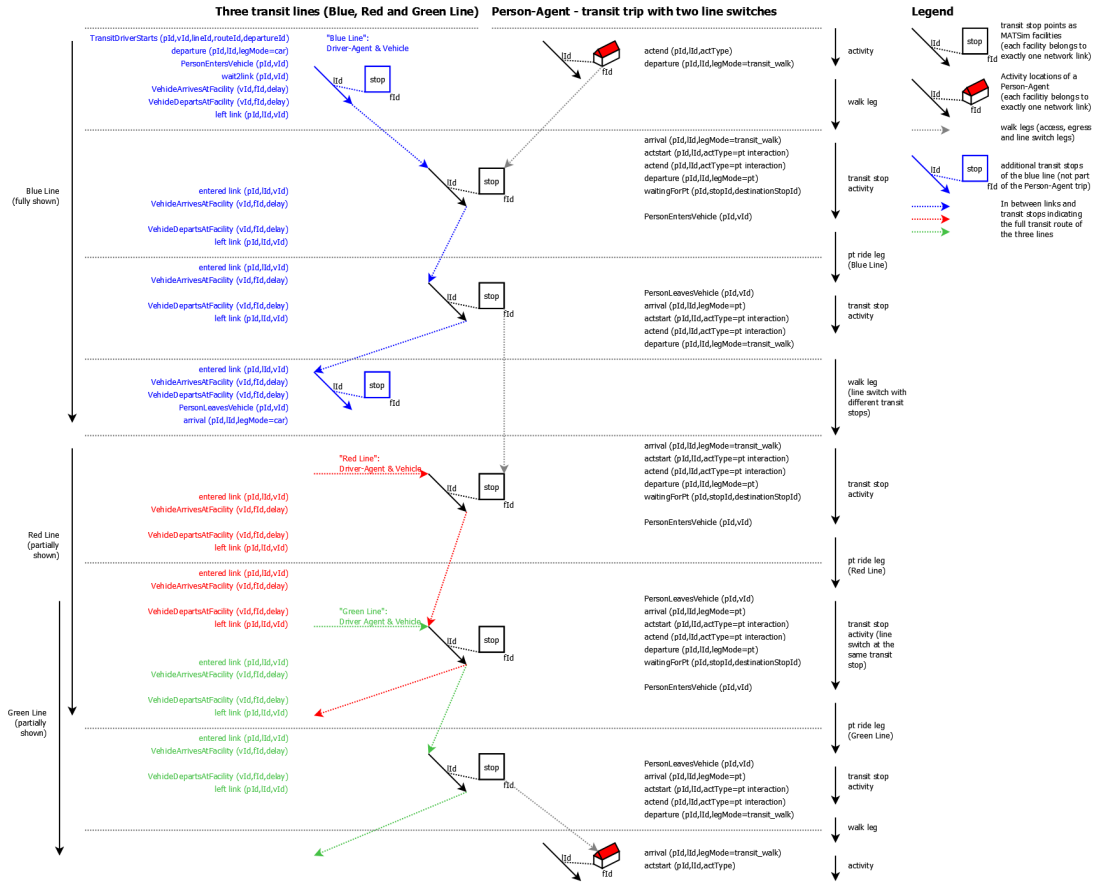


Figure 16.2: Events for a complex public transit trip.

[1] From the command line:

```
java -Xmx512m -cp /path/to/matsim.jar
      org.matsim.pt.utils.TransitScheduleValidator
      /path/to/transitSchedule.xml /path/to/network.xml
```

16.5 Swiss Rail Raptor

A fast PT router implementation is the so-called `SwissRailRaptor`. It was included into the MATSim main repository in release 12; since release 13 it is the default.[2] More information can currently (May'21) be found here: `https://github.com/SchweizerischeBundesbahnen/matsim-sbb-extensions#swissRailRaptor`.

16.6 Running with small sample sizes

Running simulations with a reduced population sample leads to artifacts when public transport is used. In a simulation with a downsampled demand, network capacity is reduced accordingly to accommodate the fact that fewer private cars are on the road. Yet because 100 % of public transport vehicles must run (albeit with reduced passenger capacity) in order to make sense of the schedule, these pt vehicles may use up too much of the flow capacity.

The way out, at this point, is to reduce each PT vehicle's `passengerCarEquivalents` value (see Section 16.4.1) manually.

This, however, will lead to another artefact: Assume a link with a capacity of 900/h, i.e. 1 vehicle every 4 seconds. Now assume a flow capacity factor of 0.01, resulting in 1 vehicle every 400 seconds for that specific link. This means that when a car leaves the link, an immediately following vehicle has to wait for 400 seconds until the following vehicle can leave the link. This is called "capacity accumulation" in the code – after a vehicle has left the link, flow capacity needs to be accumulated until the next vehicle can leave.

Now if that following vehicle is a PT vehicle, it will be accordingly delayed. This leads to spurious delays, occuring when a PT vehicle has "bad" luck, and otherwise not.

A way to reduce this problem is the parameter `pcuThresholdForFlowCapacityEasing` in the `QSimConfigGroup` (currently (May'21) only available in Java, not from config file). The syntax approximately is

```
config.qsim().setPcuThresholdForFlowCapacityEasing( 0.5 );
```

The assumption is that pt vehicles will be downscaled in the vehicle description in their PCU (Passenger Car Unit) values, but cars will not. For example, in a 10% scenario, a bus with a PCU of 3 might be entered with a value of 0.3. A value of, say, 0.5 for `pcuThresholdForFlowCapacityEasing` now means that all vehicles with PCU values of 0.5 or smaller do not have to wait for capacity accumulation.

There is some discussion at the corresponding pull request[3] and in the QA.[4]

16.7 Possible Improvements

While the ability to simulate public transport was a big advance for MATSim, several elements could be improved:

- The data model (and thus, the simulation) does not fully support some real world transit lines: for example, circular lines with no defined start and end cannot be easily modeled. Also, split trains cannot be modelled: In MATSim, only one direction can go through; for the other, the

[2] If needed, the previous (much slower) transit router can be configured from the config by

```
<module name="transit" >
   <param name="routingAlgorithmType" value="DijkstraBased"/>
   ...
</module>
```

[3] `https://github.com/matsim-org/matsim-libs/pull/55`

[4] `https://github.com/matsim-org/matsim-code-examples/issues/395`

synthetic traveller would have to change trains. Some bus or train lines also have stops where only boarding or alighting the vehicle is allowed, but not both (e.g., overnight trains with sleeper cabins). At the moment, MATSim always allows boarding and alighting at stops, leading to agents e.g., using a train with sleeper cabins for a short trip; in reality, they would be denied boarding without a reservation for a longer trip.

- The public transport router available and used by MATSim by default is strictly schedule-based. It assumes vehicles can keep up with the schedule and that enough passenger capacity is provided. In some regions, where transit is chronically delayed and overcrowded, MATSim's PT router will consistently advise agents to use routes that will perform badly in the simulation. Additional feedback from the simulation back to the router, as also done in the MATSim private car router, is available as so-called events-based public transit router (Chapter **??**), but is not integrated into the main code base.

- Currently, public transit fares can be modelled using the alternative specific constant, monetaryDistanceRate, and dailyMonetaryConstant in the scoring parameters. Whereas the former two are useful to model single tickets, the latter can be used to model e.g. monthly subscription tickets. Note that the alternative specific constant is applied to each public transit trip, i.e. it is paid only once no matter how many transfers there are. However, this is not a practical model of many real world fare systems, especially if fares differ by public transit mode, operator, route chosen or combinations thereof. The heterogeneity of fare systems worldwide and the lack of computer readable fare data are major obstacles here. GTFS (General Transit Feed Specification) feeds can contain fare information, but more often than not those fiels are not filled out. Another issue is the decision between single trip tickets and monthly or yearly subscription tickets. Usually MATSim models contain only one typical workday, but no weekends. This makes it difficult to determine whether a subscription ticket for a month would be cheaper than single tickets for each trip. There is work by ETH Zürich on fares in Switzerland.[5]

16.8 Applications

Public transport simulation has been used in many applications of MATSim world-wide. The following list highlights some of these applications, pinpointing their special public transport simulation features.

- Berlin: the Berlin scenario (see Chapter **??**) was one of the first real applications using public transport simulation in MATSim. The road and rail network, as well as the full transit schedule, was converted from a VISUM model. It is still one of the few known models where bus and tram lines share a common network with private car traffic, enabling full interaction between private and public vehicles (like transit vehicles) getting stuck and delayed in traffic jams.

- Switzerland: Senozon AG maintains a model of Switzerland containing the full timetable of all buses, trams, trains, ships, and even cable cars, in the Swiss alps. The schedule data is retrieved from the official timetable, available in a machine-readable format called "HAFAS (HaCon Fahrplan-Auskunfts-System) raw data format".

- Singapore: The model of Singapore (see Chapter **??**) makes heavy use of public transport, and continually pushes the boundaries of what is currently possible to simulate. Due to the very large number of buses on Singapore's roads and strong demand for public transport, many extensions had to be implemented to realistically model pt in this context.

[5] https://gitlab.ethz.ch/ivt-vpl/populations/ch-pt-utils, https://doi.org/10.3929/ethz-b-000342816

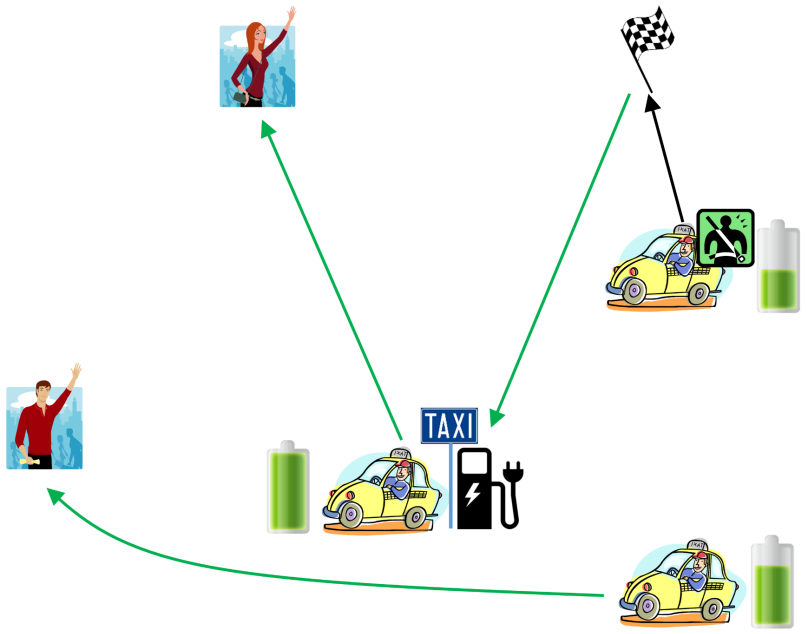
- Minibus: The minibus contribution (see Chapter **??**) added an optimization layer to public transport functionality in MATSim, allowing automatic generation of an optimized transit schedule for a specific region.
- WagonSim: In the WagonSim contribution (see Chapter **??**) public transport simulation was used to simulate rail-bound freight traffic. While the simulation was still moving around transit vehicles and letting passengers enter and leave these vehicles, the scenario had been customized so that vehicles corresponded to freight trains and passengers corresponded to actual goods being transported. Custom implementations of transit driver logic replaced vehicle capacity definition by an alternative definition, ensuring that the trains vehicles represent did not get too long or heavy. The network was constructed so that changing vehicles at stops took minimum time, corresponding to the time needed for switching wagons at freight terminals.

In addition to applications mentioned in the list above, many additional scenarios now use public transport simulation in MATSim. Importantly, the list also shows, that with some custom extensions and imagination, public transport functionality can be used for far more than "just simulating public transport"; it can be employed to solve complex problems previously handled by operations research groups.

23

Dynamic Transport Services

Author: Michal Maciejewski



Entry point to documentation:
`http://matsim.org/extensions` → dvrp

Invoking the module:
No predefined invocation. Starting point(s) under `http://matsim.org/javadoc` → dvrp → `RunOneTaxiExample` class.

Selected publications:
Maciejewski and Nagel (2013a,b, 2014); Maciejewski (2014)

23.1 Introduction

The recent technological advancements in ICT (Information and Communications Technology) provide novel, on-line fleet management tools, opening up a broad range of possibilities for more intelligent transport services: flexible, demand-responsive, safe and energy/cost efficient. Significant enhance-

ments can aid in both traditional transport operations, like regular public transport or taxis and introduction of novel solutions, such as demand-responsive transport or personal rapid transport. However, the growing complexity of modern transport systems, despite all benefits, increases the risk of poor performance, or even failure, due to lack of precise design, implementation and testing.

One solution is to use simulation tools offering a wide spectrum of possibilities for validating transport service models. Such tools have to model, in detail, not only the dynamically changing demand and supply of the relevant service, but also traffic flow and other existing transport services, including mutual interactions/relations between all these components. Although several approaches have been proposed (e.g., Regan et al., 1998; Barcelo et al., 2007; Liao et al., 2008; Certicky et al., 2014), as far the author knows, no existing solutions provide large-scale microscopic simulation that include all the components above.

23.2 DVRP Contribution

To address the problem above, MATSim's DVRP (Dynamic Vehicle Routing Problem) contribution has been developed. The contribution is designed to be highly general and customizable to model and simulate a wide range of dynamic vehicle routing and scheduling processes. Currently, the domain model is capable of representing a wide range of one-to-many and many-to-many VRPs (Vehicle Routing Problems); one can easily extend the model even further to cover other specific cases (see Section 23.3). Since online optimization is the central focus, the DVRP contribution architecture allows plugging in of various algorithms. At present, there are several different algorithms available, among them an algorithm for the *Dynamic Multi-Depot Vehicle Routing Problem with Time Windows and Time-Dependent Travel Times and Costs*, analyzed in (Maciejewski and Nagel, 2012), and a family of algorithms for online taxi dispatching, studied in (Maciejewski and Nagel, 2013a,b, 2014; Maciejewski, 2014).

The DVRP contribution models both supply and demand, as well as optimizing fleet operations, whereas MATSim's core is used for simulating supply and demand, both embedded into a large-scale microscopic transport simulation. In particular, the contribution is responsible for:

- modeling the DVRP domain,
- listening to simulation events,
- monitoring the simulation state (e.g., movement of vehicles),
- finding least-cost paths,
- computing schedules for drivers/vehicles,
- binding drivers' behavior to their schedules, and
- coordinating interaction/cooperation between drivers, passengers and dispatchers.

Dynamic transport services are simulated in MATSim as one component of the overall transport system. The optimizer plugged into the DVRP contribution reacts to selected events generated during simulation, which could be: request submissions, vehicle departures or arrivals, etc. Additionally, it can monitor the movement of individual vehicles, as well as query other sources of online information, e.g., current traffic conditions. In response to changes in the system, the optimizer may update drivers' schedules, either by applying smaller modifications or re-optimizing them from scratch. Drivers are notified about changes in their schedules and adjust to them as soon as possible, including immediate diversion from their current destinations. For passenger transport, such as taxi or demand-responsive transport services, interactions between drivers, passengers and the dispatcher are simulated in detail, including calling a ride or picking up and dropping off passengers.

23.3 DVRP Model

The DVRP contribution can be used for simulating *Rich VRPs*. Compared to the classic *Capacitated VRP*, the major model enhancements are:

- one-to-many (many-to-one) and many-to-many topologies,
- multiple depots,
- dynamic requests,
- request and vehicle types,
- time windows for requests and vehicles,
- time-dependent stochastic travel times and costs, and
- network-based routing (including route planning, vehicle monitoring and diversion).

Except for the travel times and costs (discussed in Section 23.3.2), which are calculated on demand, all the VRP-related data are accessible via `VrpData`.[1] In the most basic setup, there are only two types of entities, namely `Vehicles` and `Requests`. This model, however, can be easily extended as required. For instance, for an electric vehicle fleet, specialized `ElectricVrpData` also stores information about `Chargers`. This, and other examples of extending the base VRP model, such as a model of the *VRP with Pickup and Delivery*, are available in the `org.matsim.contrib.dvrp.extensions` package.

23.3.1 Schedule

Each `Vehicle` has a `Schedule`, a sequence of different `Tasks`, such as driving from one location to another (`DriveTask`), or staying at a given location (e.g., serving a customer or waiting; `StayTask`).[2] A `Schedule` is where supply and demand are coupled. All schedules are calculated by an online optimization algorithm (see Section 23.6) representing the fleet's dispatcher. Each task is in one of the following states (defined in the `Task.TaskStatus` enum): `PLANNED`, `STARTED` or `PERFORMED`; each schedule's status is one of the following:

- `UNPLANNED`—no tasks assigned
- `PLANNED`—all tasks are `PLANNED` (none of them started)
- `STARTED`—one of the tasks is `STARTED` (this is the schedule's `currentTask`; the preceding tasks are `PERFORMED` and the succeeding ones are `PLANNED`)
- `COMPLETED`—all tasks are `PERFORMED`

In general, when modifying a `Schedule`, one can freely change and rearrange the planned tasks; those performed are considered to be read-only. For the current task, one can, for instance, change its end time, although the start time must remain unchanged. Proceeding from the current task to the next one is carried out by invoking the `Schedule.nextTask()`) method.

The execution of the current task may be monitored with a `TaskTracker`.[3] In the most basic version, trackers predict only the end time of the current task. More complex trackers also provide detailed information on the current state of task execution. `OnlineDriveTaskTracker`, for example, offers functionality similar to GPS navigation, such as monitoring the movement of a vehicle, predicting its arrival time and even diverting its path.

[1] Package `org.matsim.contrib.dvrp.data`.
[2] Package `org.matsim.contrib.dvrp.schedule`.
[3] Package `org.matsim.contrib.dvrp.tracker`.

ScheduleImpl, along with DriveTaskImpl and StayTaskImpl, is the default implementation of Schedule and offers several additional features, such as data validation or automated task handling. It also serves as the starting point when implementing domain-specific schedules or tasks (e.g., ChargeTask in the electric VRP model mentioned above).

23.3.2 Least-Cost Paths

MATSim's network model consists of nodes connected by one-way links. Because of the queue-based traffic flow simulation (Section 1.3), a link is the smallest traversable element (i.e., a vehicle cannot stop in the middle of a link). Besides links, the DVRP contribution also operates on a higher level of abstraction: paths. Each path is a sequence of links to be traversed to get from one location to another in the network, or more precisely, from the end of one link end to the end of another link.

The functionality of finding least-cost paths is available in the org.matsim.contrib.dvrp.router package. VrpPathCalculator calculates VrpPaths by means of the least-cost path search algorithms available in MATSim's core (Jacob et al., 1999; Lefebvre and Balmer, 2007).[4] Because of changing traffic conditions, paths are calculated for a given departure time. Since MATSim calculates average link travel time statistics for every 15 minutes time period by default, the 15 minutes time bin is also used for computing shortest paths.

VrpPaths are used by DriveTasks to specify the link sequence to be traversed by a vehicle between two locations. It is possible to divert a vehicle from its destination by replacing the currently followed VrpPath with a DivertedVrpPath.

To reduce computational burden, the already calculated paths can be cached for future reuse (see VrpPathCalculatorWithCache). However, when calculating least-cost paths from one location to many potential destinations, a significant speed-up can be achieved by means of least-cost tree search (see org.matsim.utils.LeastCostPathTree).

23.4 DynAgent

Contrary to the standard day-to-day learning in MATSim (but see also Section **??**), in the DVRP contribution, each driver behaves dynamically and follows orders coming continuously from the dispatcher. The DynAgent class, along with the org.matsim.contrib.dynagent package, provides the foundation for simulating dynamically behaving agents. Although created for DVRP contribution needs, DynAgent is not limited to this context and can be used in a wide range of different simulation scenarios where agent dynamism is required.

DynAgent's main concept assumes an agent can actively decide what to do at each simulation step instead of using a pre-computed (and occasionally re-computed; see **??**) plan. It is up to the agent whether decisions are made spontaneously or (re-)planned in advance. In some applications, a DynAgent may represent a fully autonomous agent acting according to his/her desires, beliefs and intentions, whereas in other cases, it may be a non-autonomous agent following orders systematically issued from the outside (e.g., a driver receiving tasks from a centralized vehicle dispatching system).

23.4.1 Main Interfaces and Classes

The DynAgent class is a dynamic implementation of MobsimDriverPassengerAgent. Instead of executing pre-planned Activitys and Legs, a DynAgent performs DynActivitys and DynLegs. The following assumptions underlie the agent's behavior:

[4] Package org.matsim.core.router.

- The DynAgent is the physical representation of the agent, responsible for the interaction with the real world (i.e., traffic simulation).
- The agent's high-level behavior is controlled by a DynAgentLogic that can be seen as the agent's brain; the DynAgentLogic is responsible for deciding on the agent's next action (leg or activity), once the current one has ended.
- Dynamic legs and activities fully define the agent's low-level behavior, down to the level of a single simulation step.

At the higher level, the DynAgent dynamism results from the fact that dynamic activities and legs are usually created on the fly by the agent's DynAgentLogic; thus, the agent does not have to plan future actions in advance. When the agent has a roughly detailed legs and activities plan, he/she does not have to adhere to it and may modify his/her plan at any time (e.g., change the mode or destination of a future leg, or include or omit a future activity).

Low-level dynamism is provided by the execution of dynamic activities and legs. As for the currently executed activity, the agent can shorten or lengthen its duration at any time. Additionally, at each time step, the agent may decide what to do right now (e.g., communicate with other agents, re-plan the next activity or leg, and so on). When driving a car (DriverDynLeg), the agent can change the route, destination or even decide about picking up or dropping off somebody on the way. When using public transport (PTPassengerDynLeg), the agent chooses which bus to get on and at which stop to exit.

Incidentally, the behavior of MATSim's default plan-based agent, PersonDriverAgentImpl, can be simulated by DynAgent, combined with the PlanToDynAgentLogicAdapter logic. This adapter class creates a series of dynamic activities and legs that mimics a given Plan of static Activity and Leg instances.

23.4.2 Configuring and Running a Dynamic Simulation

DynAgent has been written for and validated against QSim. Dynamic leg simulation requires no additional code. However, to take advantage of dynamic activities, DynActivityEngine should be used, instead of ActivityEngine. The doSimStep(double time) method of DynActivityEngine ensures that dynamic activities are actively executed by agents and that their end times can be changed.

The easiest way to run a single iteration of QSim is as follows:

1. Create and initialize a Scenario,
2. call DynAgentLauncherUtils' initQSim(Scenario scenario) method to create and initialize a QSim; this includes creating a series of objects, such as an EventsManager, DynActivityEngine, or TeleportationEngine,
3. add AgentSources of DynAgents and other agents to the QSim,
4. run the QSim simulation, and
5. finalize processing events by the EventsManager.

Depending on needs, the procedure above can be extended with additional steps, such as adding non-default engines or departure handlers to the QSim.

23.4.3 RandomDynAgent Example

The org.matsim.contrib.dynagent.examples.random package contains a basic illustration of how to create and run a scenario with DynAgents. To highlight differences with plan-based agents, in this example 100 dynamic agents travel randomly (RandomDynLeg) and perform random duration activities (RandomDynActivity).

High-level random behavior is controlled by `RandomDynAgentLogic`, that operates according to the following rules:

1. Each agent starts with a `RandomDynActivity`; see the `computeInitialActivity(DynAgent agent)` method.
2. Whenever the currently performed activity or leg ends, a random choice on what to do next is made between the following options: (a) stop being simulated by starting a deterministic `StaticDynActivity` with infinite end time, (b) start a `RandomDynActivity`, or (c) start a `RandomDynLeg`; see the `computeNextAction(DynAction oldAction, double now)` method.

The lower level stochasticity results from random decisions being made at each consecutive decision point. In the case of `RandomDynLeg`, each time an agent enters a new link, he or she decides whether to stop at this link or to continue driving; in the latter case, the subsequent link is chosen randomly; see the `RandomDynLeg(Id<Link> fromLinkId, Network network)` constructor and the `movedOverNode(Id<Link> newLinkId)` method. As for `RandomDynActivity`, at each time step the `doSimStep(double now)` method is called and a random decision is made on the activity end time.

Following the rules specified in Section 23.4.2, setting up and running this example scenario is straightforward. `RandomDynAgentLauncher` reads a network, initializes a `QSim`, then adds a `RandomDynAgnetSource` to the `QSim`, and finally, launches visualization and starts simulation. The `RandomDynAgentSource` is responsible for instantiating 100 `DynAgent`s that are randomly distributed over the network. The simulation ends when the last active agent becomes inactive.

23.5 Agents in DVRP

Realistic simulation of dynamic transport services requires a proper model of interactions and possible collaborations between the main actors: drivers, customers (often passengers) and the dispatcher. By default, drivers and passengers are simulated as agents, while the dispatcher's decisions are calculated by the optimization algorithm (see Section 23.6). This, however, is not the only possible configuration. One may simulate, for example, a decentralized system with a middleman as dispatcher rather than the fleet's manager.

23.5.1 Drivers

A driver is modeled as a `DynAgent`, whose behavior is controlled by a `VrpAgentLogic` that makes the agent follow the dynamically changing `Schedule`.[5] As a result, all changes made to the schedule are visible to and obeyed by the driver. Whenever a new task is started, the driver logic (using a `DynActionCreator`) translates it into the corresponding dynamic action. Specifically, a `DriveTask` is executed as a `VrpLeg`, whereas a `StayTask` is simulated as a `VrpActivity`. Both `VrpLeg` and `VrpActivity` are implemented so that any change to the referenced task is automatically visible to them. At the same time, any progress made while carrying them out is instantly reported to the task tracker.

23.5.2 Passengers

To simulate passenger trips microscopically, passengers are modeled as `MobsimPassengerAgent` instances. As part of the simulation, they can board, ride and, finally, exit vehicles. In contrast to the drivers, they may be modeled as the standard MATSim agents, each having a fixed daily plan consisting of legs and activities.

[5] Package `org.matsim.contrib.dvrp.vrpagent`.

Interactions between drivers, passengers and the dispatcher, such as submitting PassengerRequests or picking up and dropping off passengers, are coordinated by a PassengerEngine[6]. Requests may be immediate (*as soon as possible*) or made in advance (*at the appointed time*). In the former case, a passenger starts waiting just after placing the order;in the latter case, the dispatched vehicle my arrive at the pickup location before or after the designated time, which means that either the driver or the customer, respectively, will wait for the other to come. To ensure proper coordination between these two agents, the pickup activity performed by the driver must implement the PassengerPickupActivity interface.

23.6 Optimizer

Since demand and supply are inherently stochastic, the general approach to dealing with dynamic transport services consists of updating vehicles' schedules in response to observed changes (i.e., events). This can be done by means of re-optimization procedures that consider all requests (within a given time horizon) or fast heuristics focused on small updates of the existing solution, rather than constructing a new one from scratch. Usually, re-optimization procedures give higher quality solutions compared to ocal update heuristics; however, when it comes to real-world applications, where high (often real-time) responsiveness is crucial, broad re-optimization may be prohibitively time-consuming.

In the most basic case, an optimizer implements the VrpOptimizer interface[7], that is, implements the following two methods:

- requestSubmitted(Request request)—called on submitting request; in response, the optimizer either adapts vehicles' schedules so that request can be served, or rejects it.
- nextTask(Schedule<? extends Task> schedule)—called whenever schedule's current task has been completed and the driver switches to the next planned task; this is the last moment to make or revise the decision on what to do next.

This basic functionality can be freely extended. Besides request submission, one may, for example, consider modifying or even canceling already submitted requests. Another option is monitoring vehicles as they travel along designated routes and reacting when they are ahead of/behind their schedules. Such functionality is available by implementing VrpOptimizerWithOnlineTracking's nextLinkEntered(DriveTask driveTask) method, which is called whenever a vehicle moves from the current link to the next one on its path.

Last but not least, there are two ways of responding to the incoming events. They can be handled either *immediately* (*synchronously*) or *between time steps* (*asynchronously*). In the former case, schedules are re-calculated (updated or re-optimized) directly, in response to the calling of the optimizer's methods. This simplifies accepting/rejecting new requests, since the answer is immediately passed back to the caller. In the latter case, all events observed within a simulation step are recorded and then processed in batch mode just before the next simulation step begins.[8] By doing that, one can not only speed up computations significantly, but also avoid situations when, due to vehicles' inertia (e.g., an idle driver can stop waiting and depart only at the beginning of the simulation step), two or more mutually conflicting decisions could be made by the optimizer at distinct moments during a single simulation step, causing the latter to overwrite the former (not always intentional).

[6] Package org.matsim.contrib.dvrp.passenger.

[7] Package org.matsim.contrib.dvrp.optimizer.

[8] This can be achieved by using an optimizer implementing the interface org.matsim.core.mobsim.framework.listeners.MobsimBeforeSimStepListener.

23.7 Configuring and Running a DVRP Simulation

Like in within-day replanning (see Chapter **??**), dynamic transport services are typically run with the DVRP contribution as a single-iteration simulation. Setting up and running such a simulation requires carrying out the following steps:

1. Create a `Scenario` (MATSim's domain data) and `VrpData` (VRP's domain data),
2. create a `VrpOptimizer`; this includes instantiation of a least-cost path/tree calculator, e.g., `VrpPathCalculator`, and
3. call `DynAgentLauncherUtils`' `initQSim(Scenario scenario)` method to create and initialize a `QSim`; this includes creating a series of objects, such as an `EventsManager`, `DynActivityEngine`, or `TeleportationEngine`.
4. When simulating passenger services, add a `PassengerEngine` to the `QSim`; this includes instantiation of a `PassengerRequestCreator` that converts calls/orders into `PassengerRequests`; otherwise (i.e., non-passenger services), add an appropriate source of requests to the `QSim`, either as a `MobsimEngine` or `MobsimListener`.
5. Then, add `AgentSources` to the `QSim`; for the `DynAgent`-based drivers, one may use a specialized `VrpAgentSource` and provide a `DynActionCreator`.[9]
6. run the `QSim` simulation, and
7. finalize processing events by the `EventsManager`.

The `org.matsim.contrib.dvrp.run` package contains `VrpLauncherUtils` and other utility classes that simplify certain steps of the above scheme. To facilitate access to the data representing the current state of the simulated dynamic transport service, `MatsimVrpContext` provides the `Scenario` and `VrpData` objects and the current time (based on the timer of `QSim`).

The `VrpOptimizer`'s performance may be assessed either by analyzing the resulting schedules, or by processing events collected during the simulation.

23.8 OneTaxi Example

The `org.matsim.contrib.dvrp.examples.onetaxi` package contains a simple example of how to simulate on-line taxi dispatching with the DVRP contribution. In this scenario, there are ten taxi customers and one taxi driver, who serves all requests in the FIFO order. Each customer dials a taxi at a given time to get from work to home. The example is made up of six classes:

- `OneTaxiRequest`—represents a taxi request.
- `OneTaxiRequestCreator`—converts taxi calls into requests prior to submitting them to the optimizer.
- `OneTaxiOptimizer`—creates and updates the driver's schedule.
- `OneTaxiServeTask`—represents `StayTasks` related to picking up and dropping off customers.
- `OneTaxiActionCreator`—translates tasks into dynamic activities and legs.
- `OneTaxiLauncher`—sets up and runs the scenario.

All data necessary to run the OneTaxi example is located in the `/contrib/dvrp/src/main/resources/one_taxi` directory.

[9] Package `org.matsim.contrib.dvrp.vrpagent`,

23.9 Research with DVRP

Currently, the DVRP contribution is used in several research projects. Two of them focus on on-line dispatching of electric taxis in Berlin and Poznan (Maciejewski and Nagel, 2013a,b, 2014; Maciejewski, 2014). Another project deals with design of demand-responsive transport, where DVRP has been applied to the case of twin towns, Yarrawonga and Mulwala, described in Chapter **??** (Ronald et al., 2015, 2014). In a recently launched project, the DVRP contribution will be used for simulation of DRT (Demand Responsive Transport) services in three cities: Stockholm, Tel Aviv and Leuven.

The current code development focuses on increasing performance and flexibility of the implemented shortest paths search (see Section 23.3.2). An interesting future research topic, related specifically to DRT planning, is multi-modal path search, where on-demand vehicles may be combined with fixed-route buses within a single trip. Another potential research direction is adding a benchmarking functionality and standardized interfaces so that the DVRP contribution could serve as a testbed for the *Rich VRP* optimization algorithms.

45

How to Write Your Own Extensions and Possibly Contribute Them to MATSim

Author: Michael Zilske, Kai Nagel

Notes

Documentation for the concepts described in this chapter can be found under `http://matsim.org/javadoc` → main distribution, by going to the corresponding class and interface documentation entries. These should also point to examples.

For programming against the MATSim API, we recommend forking or cloning `https://github.com/matsim-org/matsim-example-project` as a starting point. This should also clarify how MATSim can be used as an Maven plug-in.

Code examples can be found at `https://github.com/matsim-org/matsim-code-examples`. Evidently, this can also be cloned/forked, but it can also be used via the browser.

In general, we recommend to *not* clone or even fork the material under `https://github.com/matsim-org/matsim-libs`, since it is more robust in many ways to pull it via the Maven infrastructure, either as release (stable), or as development snapshot.

45.1 Introduction

The three main elements of the MATSim cycle, execution, scoring, and replanning (Section 1.4), operate on what is essentially an in-memory, object-oriented data base of `Person` objects (Raney and Nagel, 2006). The following three elements are the main elements to configure MATSim:

Execution The mobsim can be replaced, either by an internally available alternative, or by a fully external mobsim.

Scoring The scoring can be replaced, by possibly giving each individual agent a different recipe to compute its score.

Replanning Arbitrary implementations of type `PlanStrategy` can be added to the replanning; these either generate new plans from scratch or mutate existing ones, or they select between plans.

The simulation's behavior can be further configured by using `ControlerListener`s.

The mobsim generates a stream of events. These are primarily used in two places:

- The scoring uses events to track each agent's success at executing its plan, and computes the scoring value based on this.
- `PlanStrategy` modules use events to build approximate models of the world in which they operate. For example, the router obtains time-dependent expected link travel times from a `TravelTime` object, which in turn listens to link enter and link exit events.

Additionally, one can write any sort of event handlers for analysis during the iterations, or after a run by evaluating the events file.

Some modules are so large that fully replacing them in order to adapt the simulation system to one's needs is too much work. These are, in particular,

- the QSim, which is the default implementation of the `Mobsim` interface, and
- the router.

To address this issue, it is possible to add additional executable code into the execution flow of the QSim by `MobsimListener`s in a similar way as this is possible with the `ControlerListener`s mentioned above. The router, in contrast, is most importantly configured by replacing the definition of the generalized travel cost, or by adding/replacing the routing modules for specific modes.

45.2 Preparation: Scripts-in-Java

As stated in Sec. 3.2.3, we recommend writing "scripts in Java". The syntax, again, is roughly:

```
... main( ... ) {
    // construct the config object:
    Config config = ConfigUtils.xxx(...) ;
    config.xxx().setYyy(...) ;
    ...

    // load and adapt the scenario object:
    Scenario scenario = ScenarioUtils.loadScenario( config ) ;
    scenario.getXxx().doYyy(...) ; // (*)
    ...

    // load and adapt the controler object:
    Controler controler = new Controler( scenario ) ;
    controler.doZzz(...) ; // (**)
    ...

    // run the iterations:
    controler.run() ;
}
```

Again, as a starting point, clone/fork `https://github.com/matsim-org/matsim-example-project`, and for coding examples, look at `https://github.com/matsim-org/matsim-code-examples`.

Evidently, such a script-in-Java can be used to

- modify the `Config`, by adding or changing config entries
- modify the `Scenario`, by adding or changing scenario entries, and to
- modify the `Controler`, by a syntax explained below.

45.3 MATSim Extension Points: General

This chapter (Chapter 45) describes what could be called the SPI (Service Provider Interface) of MATSim. Historically, the main entry point for writing a MATSim extension had been to literally extend (in the Java sense, i.e. to inherit from) the `Controler` class. Essentially, one would override the methods calling the mobsim, the scoring, and/or the replanning, as explained in Section 45.1. This is now discouraged. While this pattern worked when a each member of the team was working on extending the MATSim core by a different aspect, it fails when it comes to integrating those aspects to a single product: It is not possible to straightforwardly combine a `PublicTransportControler`, an `EmissionsControler`, a `RoadPricingControler` and an `OTFVisControler`, if one wants to combine them to visualize the emissions of buses on toll roads. Also see Section **??**. In contrast, with the injection approach described in Sec. 45.6 it is possible to combine modules while keeping their respective dependencies independent.

45.4 Extension Points Related to Scenario

45.4.1 Customizable, ObjectAttributes and Attributable

MATSim operates on data types such as links, nodes, persons, or vehicles. Many of these data types have attributes, such as free speed (for links) or coordinates (for nodes). Rather often, one would like additional information for certain data types, such as "slope" for links, or "age" for persons. In order to not modify the Java data types every time this becomes necessary, but still allow experimentation, there are three approaches to address this situation. In principle, learning the most recent of those, `Attributable`, should be sufficient. However, there is some older code that has not yet been modernized, and for that one needs to know the older approaches.

Attributable Many MATSim data types implement this interface, including some that do not have an ID, and it is planned that eventually all MATSim scenario data types will implement it.

The use of the `Attributable` interface is quite straighforward: Data types implementing this interface allow a syntax of type

```
<dataType>.getAttributes().putAttribute( String attribute, Object value) ;
```

The content can be retrieved by

```
Object value = <dataType>.getAttributes().getAttribute( String attribute ) ;
```

For a working example see RunPersonAttributesExample in matsim-code-examples.

These additional attributes will be written into the XML file, and read back in. For this, there are `AttributeConverter`s attached to the corresponding writers and readers. They have default implementations for the standard data types such as strings, numbers, and enums, but if a more complex user defined data type is to be used as `Attribute`, then the user also needs to supply the corresponding `AttributeConverter`.

ObjectAttributes Besides `Customizable`, a helper container called `ObjectAttributes` is available. It essentially attaches arbitrary additional information to objects *that have an ID*, by a syntax of type

```
attribs.putAttribute( id, attribName, attribValue ) ;
```

where `id` is the object's ID, `attribName` is the name (type) of the attribute to be stored (e.g., "age"), and `attribValue` is the value of the attribute (e.g., "24").

The package provides readers and writers for such attributes. It is thus possible for additional code to, say, generate additional attributes by preprocessing, write them to file, and read them back for every

run. That approach is used, for example, by the Gauteng scenario (Chapter **??**) to pre-allocate e-tag ownership to persons.

Please check the documentation of ObjectAttributes (see http://matsim.org/javadoc → main distribution) for more details and pointers to examples.

Clearly, ObjectAttributes cannot be used for data types that do not have an ID. This is, for example, the case with plans, activities, or legs, which are contained inside a data type with an ID (the person data type), but which do not possess an ID of their own and are therefore not addressable by ObjectAttributes.

Customizable Some MATSim data types implement the Java interface Customizable, to which additional material can be attached by a syntax of type

```
<dataType>.getCustomAttributes.put("myAttribName",myAttribValue) ;
```

This is similar to Attributable, except that the additional information is not written to file. The decision to add Attributable rather than to modify Customizable was made because in order to write such an additional object to file and read it back in it is necessary to have a method that converts the objects to strings and back.

For additional information, see the Customizable interface under http://matsim.org/javadoc → main distribution.

45.4.2 Additional Scenario Elements

The object-oriented, in-memory database which holds the Person objects with their plan memories is accessible via the Population interface. The Network interface gives access to the traffic network graph, consisting of links and nodes. There is a TransitSchedule interface which represents the public transit schedule. Your own modeling tasks may need an additional data container like these. We call them scenario elements.

Scenario is the interface which ties all scenario elements together. You can add your own named scenario element to Scenario, by a syntax of type

```
final MyScenarioElement myScenarioElement = new MyScenarioElement();
scenario.addScenarioElement( MyScenarioElement.NAME, myScenarioElement );
```

and retrieve it by a syntax of type

```
MyScenarioElement mm = (MyScenarioElement) scenario.getScenarioElement( MyScenarioElement.NAME ) ;
```

See RunScenarioElementExample in matsim-code-examples for an example.

45.5 Events

The mobsim moves the agents around in the virtual world according to their plans and within the bounds of the simulated reality. It documents their moves by producing a stream of events. Events are small pieces of information describing the action of an object at a specific time. Examples of such events are (also see Figure 2.4):

- An agent finishes an activity.
- An agent starts a trip.
- A vehicle enters a road segment.
- A vehicle leaves a road segment.
- An agent boards a public transport vehicle.

- An agent arrives at a location.
- An agent starts an activity.

Each event has a timestamp, a type, and additional attributes required to describe the action like a vehicle id, a link id, an activity type or other data. In theory, it should be possible to replay the mobsim just by the information stored in the events. While a plan describes an agent's intention, the stream of events describes how the simulated day actually was.

As the events are so basic, the number of events generated by a mobsim can easily reach a million or more, with large simulations even generating more than a billion events. But as the events describe all the details from the execution of the plans, it is possible to extract essentially any kind of aggregated data one is interested in. Practically all analyses of MATSim simulations make use of events to calculate some data. Examples of such analyses are the average duration of an activity, average trip duration or distance, mode shares per time window, number of passengers in specific transit lines and many more.

The scoring of the executed plans makes use of events to find out how much time agents spend at activities or for traveling. Some replanning modules might make use of events as well: The router for example can use the information contained in events to figure out which links are jammed at certain times and route agents around that jam when creating new plans.

Handling Events MATSim extensions can watch the mobsim by interpreting the stream of events. This is done by implementing the `EventHandler` interface and registering the implementation with the framework. The lifecycle of an `EventHandler` can be chosen by the developer. Normally, an `EventHandler` lives as long as the simulation run. It is notified before the beginning of each new iteration so that its state can be reset to listen to a new iteration.

See RunEventsHandling in matsim-code-examples for several pointers to coding examples.

Producing Your Own Events One can extend the MATSim event model by extending the `Event` class to define own event types. Events can be produced from all places in the code which are executed during the running mobsim, and in particular from other `EventHandler` instances. Assume for example you want to analyze left-turns. A good starting point would be to specify a `LeftTurnEvent` class, and produce an instance of this class whenever a vehicle does a left-turn. You may do this from a class which is a `LinkLeaveEventHandler` as well as a `LinkEnterEventHandler`. A `LinkLeaveEvent` is produced every time a vehicle leaves a road segment, and a `LinkEnterEvent` is produced when it enters the next road segment. Pairing each `LinkLeaveEvent` with the next `LinkEnterEvent` for the same vehicle gives a model for a vehicle crossing a node. At this point, your code would look at the road network model to determine if this was a left-turn, and if so, produce a `LeftTurnEvent`.

See the `RainOnPersonEvent` in RunCustomScoringExample in matsim-code-examples for an example. (In that particular example, the rain event is caught in the scoring function, which is, in fact, not totally obvious.)

45.6 Configuring the `Controler` (= inject syntax)

`Controler` is the main user-facing class of MATSim. It used to be common in the MATSim community to inherit from it, but this made core maintenance difficult and is in consequence now no longer possible, i.e. the `Controler` class is now `final`. Please do not start from old examples that use such inheritance from `Controler`; they are probably also deprecated in other ways. Rather, use the binding and injection syntax described in the following.

The standard approach for configuring the Controler approximately is

```
controler.addOverridingModule( new AbstractModule(){
   @Override public void install() {
      this.<tab completion> ...
   }
} ) ;
```

An easy one is

```
      this.bindMobsim().to...
```

This replaces the mobsim.

Since this will not be a frequent use case, let us turn to another one:

```
      this.addEventHandlerBinding().to... ;
```

This adds a new event handler to the already existing ones.

Possibilities to continue include

```
...to(MyHandler.class) ;
...toInstance(new MyHandler(...)) ;
...toProvider(new MyProvider(...)) ;
```

These options will be described in the following.

45.6.1 Binding a Class (= an Implementation Type)

The simplest variant is the first, marked by the fact that it is also the shortest. MyHandler then approximately looks like

```
class MyHandler implements XxxEventHandler {
   private final Scenario scenario ;
   @Inject MyHandler( Scenario scenario, ... ) {
      this.scenario = scenario ;
      ...
   }
   ...
}
```

Note the @Injected constructor. All arguments of the constructor (in this case: Scenario) need to be bound by other binding statements. The MATSim logfile gives a list of bound objects. Also note that the constructor can be package-protected even when the class is not, making the class uninstantiable by outside users, which may often be desired.

Besides injecting variables via the constructor, it is also possible to inject them directly:

```
class MyHandler implements XxxEventHandler {
   @Inject Scenario scenario ;
   ...
}
```

This syntax is less flexible then injection of the constructor, but more concise.

45.6.2 Binding an Instance

Instead of a class, one can bind an instance, in the example denoted by new MyHandler(...), but it can also be a variable referring to an instance generated earlier. This is less flexible in terms of injection, but has the advantage that it is closest to what many people may be used to, since it just generates an instance in the standard way. It is also perfectly fine for prototype development; the conversion into using injected variables can still be done later, when other people possibly start using the newly implemented functionality.

45.6.3 Binding a Provider (not important to get started)

The third variant that is used frequently in MATSim is to bind a Provider, which is a creational class similar to a factory. The syntax approximately is

```
class MyProvider implements Provider<XxxHandler> {
   private Scenario scenario ;
   @Inject MyProvider( Scenario scenario, ... ) {
      this.scenario = scenario ;
      ...
   }
   @Override public XxxHandler get() }
      ...
      return xxxImpl ;
   }
   ...
}
```

This allows to specifically construct the injected class as needed. This is often used in the MATSim core code to configure injected classes depending on what is defined in the config. Advantages over conventional factories/builders include:

- The creational syntax is standardized to the get() method.
- The creational method has no argument, obliterating the need to define a correct list of necessary arguments.
- Since the constructor is injected, it can have as arguments arbitrary bound material. This also means that this argument list can be modified later without having to touch calling code.

45.6.4 MATSim default bindings

The MATSim default bindings can be found from ControlerDefaultsModule, see matsim.org/javadoc.

Also see https://google.github.io/guice/api-docs/latest/javadoc/index.html?com/google/inject/Binder.html, in particular "consult ... examples", which states that it is easier to look at examples than trying to understand the injection code itself.

45.6.5 Advantages of the injection framework

Standard dependency injection works via constructors:

```
ModuleA moduleA = new ModuleA(...) ;
ModuleB moduleB = new ModuleB( moduleA, ...) ;
```

This is an entirely practical and even preferrable approach in situations where it works. However, for a multi-person project it has the disadvantages that modules need to be generated in a certain sequence, and that changing module dependencies means refactoring all constructors of that module, even if these constructors sit in other users' repositories.

The dependency framework removes these restrictions:

- Dependencies of a module on other modules are provided by the inject syntax. In consequence, changing such dependencies does not change the code of persons using the module, as long as the module has dependencies that are tyically bound also in other persons' code (such as the MATSim defaults).
- The injection framework sorts out module dependencies internally, a bit similar to the Java compiler that sorts out class dependencies. That is, the injection framework constructs a module dependency graph, then starts with modules without dependencies, next constructs modules that only depend on the ones already built, etc. This will, evidently, only work as long as con-

struction is possible at all, i.e., , there are no circular dependencies on implementations. Circular dependencies that depend only on references, in contrast, are resolved by the framework.

45.6.6 Pre-configured MATSim extension points

Arbitrary material can be bound with the above syntax, and re-used via the `@Inject` statement. This is particularly useful for code that extends MATSim, which is not constrained in any way to use the injection framework for its own purposes.

In order to make the default MATSim extension points more easily accessible, they are presented as specific methods. Some of them are:

```
// unnamed adders:
this.addControlerListenerBinding()... ;
this.addEventHandlerBinding()... ;
this.addMobsimListenerBinding()... ;
// named adders:
this.addPlanSelectorForRemovalBinding(selectorName)... ;
this.addPlanStrategyBinding(selectorName)... ;
this.addTravelTimeBinding(mode)... ;
this.addTravelDisutilityFactoryBinding(mode)... ;
// setters:
this.bindLeastCostPathCalculatorFactory()... ;
this.bindScoringFunctionFactory()... ;
this.bindMobsim()... ;
```

In addition, there are the convenience methods

```
this.bindNetworkTravelTime()... ;
this.bindCarTravelDisutilityFactory()... ;
```

Note that in some situations a creational class (factory) is bound rather than the class itself. This can lead to the somewhat confusing situation that there is a `Provider` that creates such a factory. This variant is needed when the creational method in the factory takes an argument itself, as, e.g., , in

```
ScoringFunction createScoringFunction( Person person ) { ... } ;
```

This happens in particular when the created objects on MATSim data objects, such as `Person`s or `Link`s.

45.6.7 ControlerListener: Handling Controler Events

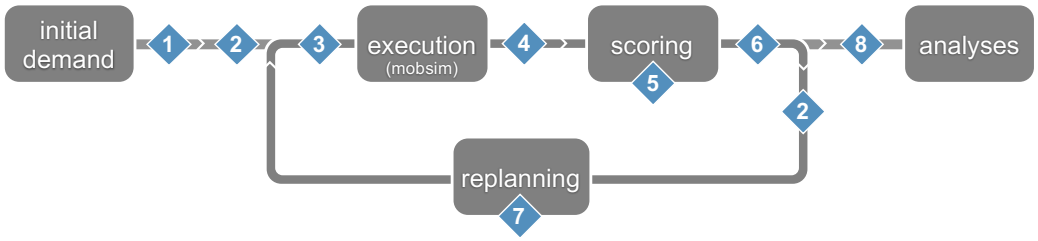
`ControlerListeners` are called at the transitions of the MATSim loop (Figure 45.1), where so-called `ControlerEvents` are fed to the listeners.

The following `ControlerListeners` are currently available: `StartupListener`, `IterationStartsListener`, `BeforeMobsimListener`, `AfterMobsimListener`, `ScoringListener`, `IterationEndsListener`, `ReplanningListener`, and `ShutdownListener`.

A sample listener might look as follows.

```
public class MyControlerListener implements StartupListener {
   @Override
   public void notifyStartup(StartupEvent event) {
      ...
   }
}
```

`ControlerListeners` are called in undefined order, meaning that `AControlerListener` may only rely on the computation of `BControlerListener` if `BControlerListener` makes that computation in an earlier transition. For instance, if `BControlerListener` is a `StartupListener` and loads data into a `Map` on start-up, `AControlerListener` can be an `IterationStartsListener` and use that `Map`. But do not write two `IterationStartsListeners` where the first puts some data into a `Map` and the second expects to find it there, they may be called in any order.



Controler Events:

1 Simulation Starts ("Startup")
2 Iteration Starts
3 Before Mobsim
4 After Mobsim
5 Scoring
6 Iteration Ends
7 Replanning
8 Simulation Ends ("Shutdown")

Figure 45.1: Controler events.

Please check the documentation of `ControlerListener` (see `http://matsim.org/javadoc` → main distribution) for more details and RunControlerListenerExample in matsim-code-examples for examples.

45.6.8 Mobsim Listener

A `MobsimListener` is called in each mobsim timestep. This can, for example, be used to produce a custom event which is not triggered by another event. For example, if you wanted to include a model of weather conditions into the simulation, you could use this extension point to decide in every time step if it should start or stop raining on a certain road segment, and produce custom events for this. You would then calculate rain exposure per agent by adding an `EventHandler` which handles `LinkEnterEvent`, `LinkLeaveEvent` and your custom rain events.

Note that `EventHandler` and `MobsimListener` instances may be run in parallel by the framework. It is generally not safe to share state between them. The framework guarantees that the methods of an `EventHandler` instance are called sequentially, but two different instances may run on different threads of execution. Access to shared data must be synchronized externally. Whenever possible, different `EventHandler` instances should only communicate through events.

See `http://matsim.org/javadoc` → main distribution → `MobsimListener` for more details and RunMobsimListenerExample in matsim-code-examples for an example.

45.6.9 TripRouter

The `TripRouter` is a service object providing methods to generate *trips* between locations, given a (main) mode, a departure time and a `Person`. A trip is a sequence of plan elements representing a movement. It typically consists of a single leg (`Leg` object), or of a sequence of legs with "*stage activities*" in between. For instance, public transport trips contain `pt interaction` activities, representing changes of vehicles in public transport trips.

Using the Router A `TripRouter` instance provides a few methods to work with trips, namely

- compute a route for a given mode and O-D (Origin-Destination) pair, for a `Person` with a specific departure time;

- identify the *main mode* of a trip. For instance, a trip composed of several *walk* and *pt* legs should be identified as a public transport trip;
- identify which activities are *stage activities*, and, by extension, identify the trips in the plan: A trip is the longest sequence of consecutive Legs and *stage activities*.

Please check http://matsim.org/javadoc → main distribution → TripRouter for more documentation and RunPluggableTripRouterExample in matsim-code-examples for an example.

Configuring the Router The TripRouter computes routes by means of RoutingModule instances, one of which is associated with each mode. A RoutingModule defines the way a trip is computed, and is able to identify the *stage activities* it generates.

The association between modes and RoutingModule instances is configurable. You can even provide you own RoutingModule implementations. Do this if your use-case requires custom routing logic, for instance, if you want to implement your own complex travel mode.

Please check once more http://matsim.org/javadoc → main distribution → TripRouter for documentation and RunTeleportationMobsimWithCustomRoutingExample in matsim-code-examples for an example.

45.6.10 Mobsim

Alternative mobsim in Java Besides adding MobsimListener implementations to enrich the standard mobsim, it is also possible to replace the entire mobsim by a custom implementation. A mobsim is basically a Runnable which is supposed to take a scenario and produce a stream of events. This allows you to use the co-evolutionary framework of MATSim while replacing the traffic model itself.

Please check http://matsim.org/javadoc → main distribution → Mobsim for documentation and RunOwnMobsimExample in matsim-code-examples for an example.

Alternative mobsim in another programming language Your implementation need not be written in Java. The framework includes a helper class to call an arbitrary executable which is then expected to write its event stream into a file. This pattern has been used successfully many times, see, e.g., Section **??**, or the CUDA (Compute Unified Device Architecture, a parallel computing platform and API by NVIDIA) implementation of the mobsim by Strippgen (2009). Note that we have found consistently that an external mobsim does *not* help with computing speed: Whatever is gained in the mobsim itself is more than lost again by the necessary data transfer between MATSim and the external mobsim. As of now, we cannot yet say if newer data exchange techniques, such as Google Protocol Buffers, may change the situation. Until then, the external interface should rather be seen as the option to inject a different, possibly more realistic, mobsim into MATSim.

45.6.11 PlanStrategy

Replanning in MATSim is specified by defining a set of weighted strategies. In each iteration, each agent makes a draw from this set and executes the selected strategy. The strategy specifies how the agent changes its behavior. Most generally, it is an operation on the plan memory of an agent: It adds and/or removes plans, and it marks one of these plans as selected.

Strategies are implementations of the PlanStrategy interface. The two most common cases are:

- Pick one plan from memory according to a specified choice algorithm.
- Pick one plan from memory at random, copy it, mutate it in some specific aspect, add the mutated plan to the plan memory, and mark this new plan as selected.

The framework provides a helper class which can be used to implement both of these strategy templates. The helper class delegates to an implementation of PlanSelector, which selects a plan from memory, and to zero, one or more implementations of PlanStrategyModule, which mutate a copy of the selected plan.

The maximum size of the plan memory per agent is a configurable parameter of MATSim. Independent of what the selected PlanStrategy does, the framework will remove plans in excess of the maximum from the plan memory. The algorithm by which this is done is another implementation of PlanSelector and can be configured.

The four most commonly used strategies shipped with MATSim are:

- Select from the existing plans at random, which are weighted by their current score.
- Mutate a random existing plan by re-routing all trips.
- Mutate a random existing plan by randomly shifting all activity end times backwards or forwards.
- Mutate a random existing plan by changing the mode of transport and re-routing one or more trips or tours.

Routes are computed based on the traffic conditions of the previous iteration, which are measured by means of an EventHandler. Using the same pattern, your own PlanStrategy can use any data which can be computed from the mobility simulation. The source code of the standard PlanStrategy implementations can be used as a starting point for implementing custom behavior.

Re-routing as a building block of many replanning strategies is a complex operation by itself. It can even be recursive: For example, finding a public transport route may consist of selecting access and egress stations as sub-destination, finding a scheduled connection between them, and finding pedestrian routes between the activity locations and the stations. With the TripRouter interface, the framework includes high-level support for assembling complex modes of transport from building blocks provided by other modules or the core.

Please check http://matsim.org/javadoc → main distribution → PlanStrategy for documentation, and RunPluggablePlanStrategyInCodeExample in matsim-code-examples, RunPluggablePlanStrategyFromFileExample, and RunWithMultithreadedModule for examples.

45.6.12 Scoring

The parameters of the default MATSim scoring function (Chapter 5) are configurable. The code, which maps a stream of mobsim events to a score for each agent, is placed behind a factory interface and replaceable. However, replacing it means replacing the entire utility formulation. For instance, a module which simulates weather conditions would probably calculate penalties for pedestrians walking in heavy rain, and the Cadyts (Chapter **??**) calibration scheme already uses utility offsets in its formulation. A modeler who wishes to compose a scoring function from the Charypar-Nagel utility, the rain penalty and the calibration offset needs to do this manually, in code, accessing the code of all three modules contributing to the score and (for instance) summing up their contributions.

Keep in mind that score and replanning are not inherently coupled or automatically consistent with each other. Consider a scoring function which penalizes left-turns. This is straight-forward to program: You would iterate over every route an agent has taken. Looking at the Network, you would calculate for each change of links if you consider it a left-turn, and if so, add a (negative) penalty to the score. However, this would not by itself lead to a solution where routes are distributed according to this scoring function. The reason is that the default replanning only proposes fastest routes, in other words, least-cost paths with respect to travel time. By default, the plan memory of an agent will only ever contain routes which have in one iteration been a fastest route. The behavior of the router is, in this case, inconsistent with the utility formulation.

Please check http://matsim.org/javadoc → main distribution → ScoringFunction for documentation, and RunCustomScoringExample in matsim-code-examples, RunIndividualizedScoringExample, RunScenarioWithCustomScoring and RunOwnMoneyScoringExample for examples.

45.7 Additional Config Groups

The configuration of a MATSim run is a grouped list of key-value pairs, stored in XML format in the config file (see Section 45.7).

At runtime, the entire configuration is stored in an instance of Config, from which instances of ConfigGroup can be accessed by their name. Config groups that are not in the main distribution need to be explicitly loaded; an approximate example is the following:

```
MyExternalConfigGroup myConfig
   = ConfigUtils.addOrGetModule(controler.getConfig(), MyExternalConfigGroup.class);
```

The author of an extension can subclass the ConfigGroup class to provide named accessors for the parameters. A possibly better way is to subclass from ReflectiveConfigGroup, which you can use if you want to define the mapping of named parameters to accessors using Java annotations.

See RunReflectiveConfigGroupExample in matsim-code-examples for an example.

45.8 MATSim extensions

Many MATSim extensions can now also be added via the inject framework. The syntax typically looks as follows:

```
   controler.addOverridingModule( new AbstractModule(){
      @Override public void install() {
         this.install( new XyzModule() ) ;
      }
   } ) ;
```

where Xyz often has some relation to the name of the extension.

If the extension is part of the main distribution, this will be it. Otherwise, the extension also needs to be added into the pom.xml file in order to make Maven load it. See, e.g. RunRobotaxiExample in matsim-code-examples, RunMinibusExample, and RunTransitWithOtfvisExample.

This is also the place where the additional config sections according to Sec. 45.7 become most important, since it often makes sense to configure the extension module from a config section rather than entirely in free-form Java code.

45.9 Conclusion

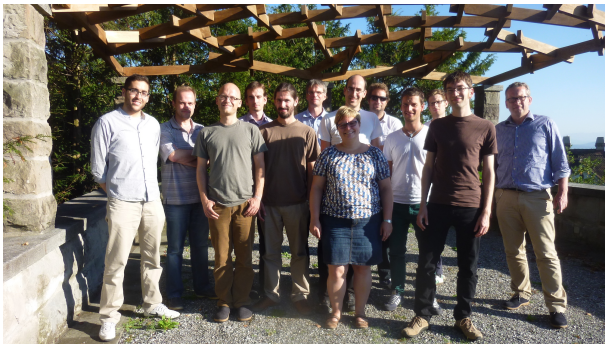
After forking or cloning the MATSim example project, one can use the example scripts-in-Java in that project as starting points for own such scripts-in-Java. These can use all the facilities descripted in this chapter, including the possibility to bind and then use arbitrary own material.

Also, existing MATSim extensions can be used in this way. In particular, it is very straightforward to include such extensions as additional Maven dependencies into the pom.xml file and then use them in this way.

44

Organization: Development Process, Code Structure and Contributing to MATSim

Authors: Marcel Rieser, Andreas Horni, Kai Nagel



This chapter describes how new functionality enters MATSim. It describes the MATSim team and community, the different roles existing in the MATSim project, the development drivers and processes, and the tools used for integration. The goal is to provide an overview of the development process so that one quickly finds access to the MATSim community and is able to efficiently contribute to MATSim, based on one role or another.

44.1 MATSim's Team, Core Developers Group, and Community

The **MATSim team** currently consists of three research groups and two companies:

- the VSP (VerkehrsSystemPlanung und Verkehrstelematik – The Transport Systems Planning and Transport Telematics group at TU Berlin) group at the ILS (Institut für Land- und Seeverkehr – Institute for Land and Sea Transport Systems), TU Berlin, led by Prof. Dr. Kai Nagel,
- the VPL (VerkehrsPLanung) group at the IVT (Institut für Verkehrsplanung und Transportsysteme – Institute for Transport Planning and Systems), ETH (Eidgenössische Technische Hochschule) Zürich, led by Prof. Dr. Kay W. Axhausen,
- the Mobility and Transportation Planning group at the FCL (Future Cities Laboratory), based in Singapore and led by Prof. Dr. Kay W. Axhausen, and

- Senozon AG, based at Zürich with subsidiaries in Germany and Austria, founded by former PhD (Philosophiae Doctor – Doctor of Philosophy) and research students,
- Simunto GmbH, based at Zürich, founded by a former PhD (Philosophiae Doctor – Doctor of Philosophy) and research student.

As is common in research, the university groups' composition changes frequently. Over the last decade more than 50 people, as listed earlier, contributed to MATSim.

A small group of the MATSim team defines the **MATSim core developers group**, maintaining MATSim's core as defined below in Section 44.3.2.

In addition, there is a **MATSim community** composed of closely connected research groups in other cities, e.g., Stockholm, Pretoria, Poznan, and Jülich, as well as more loosely connected external users coming together, e.g., at the annual MATSim User Meeting (see Figure 44.1).

MATSim is open-source software under the GPLv2 (GNU General Public License version 2.0). You are also very welcome to contribute to the code base as described in Section 44.6. New contributors are mentored in the beginning to become familiar with the project and the coding conventions.

Source: ©Dr. Marcel Rieser, Simunto GmbH

Figure 44.1: MATSim events and community.

44.2 Roles in the MATSim Community

The MATSim community includes the following roles:

- The **MATSim user** uses the official releases or nightly builds and runs the MATSim core with the config file (Section 3.2.1). He or she does not write computer code. Part I of the book is dedicated to the MATSim user. On the web page, he or she finds relevant information in the *user's guide* section and in the user's mailing list `users@matsim.org`.[1] There is also a list of questions and answers under `http://matsim.org/faq`.

  Users should also remember to consult the files `logfileWarningsErrors.log` and `output_config.xml.gz`, as also explained in Section 2.4. The former file is an extract from `logfile.log`, but only contains the warnings and errors. The latter is a complete dump of the currently available configuration options, including comments to most options.

- The **MATSim power user** is a MATSim user with knowledge on how to use the additional modules presented in the book's Part II. He or she does *not* program but knows how to use MATSim scripts-in-Java prepared by others or her/himself, as shown in Section 3.2.3. Parts I and II of the book are helpful to the MATSim power user. Information about extensions can be found under `http://matsim.org/extensions`. Most extensions come with an example script-in-Java. Again, questions and answers are under `http://matsim.org/faq`.

- The **MATSim developer** extends MATSim by programming against the MATSim API (Section 3.2.5). He or she also finds his or her information in Part II of the book, in particular, in Chapter 45, on the web page in the Developer's Guide, and in the mailing list `developers@matsim.org`.

- There are relatively few **MATSim core developers** in the MATSim team. These persons make necessary modifications of the core (as defined in Section 44.3.2), usually after having discussed them in the issue tracker (`http://matsim.org/issuetracker`), in the MATSim committee, or at a developer meeting (see below).

44.3 Code Base

The various pieces of MATSim are delineated by Maven projects and sub-projects. The Maven layout corresponds to the layout of the Git repository.[2] Note that the Java package structure does *not* directly correspond to the Maven/Git layout.

44.3.1 Main Distribution

The "MATSim main distribution" corresponds to the "matsim" part of the Git repository. It is the part of the code that the MATSim team feels primarily responsible for. At the time of writing, the MATSim main distribution contains following packages:

- `org.matsim.analysis.*`, containing certain analysis packages that are added by default to every MATSim run.

- `org.matsim.api.*`, see Section 44.3.2.

- `org.matsim.core.*`, see Section 44.3.2.

[1] During the writing of this book, the information that had so far been contained in the User's Guide was moved to this book. Therefore, the User's Guide section on the web page is currently essentially empty, and may be removed.

[2] MATSim is currently at GitHub under `https://github.com/matsim-org/matsim-libs`. The exact path name may change in the future, e.g., because of changes at GitHub.

- org.matsim.counts.*, see Section 6.3.
- org.matsim.facilities.*, see Section 6.4.
- org.matsim.households.*, see Section 6.5.
- org.matsim.jaxb.*, containing automatically or semi-automatically generated adapter classes to read XML files using JAXB (Java Architecture for XML Binding).
- org.matsim.lanes.*, see Chapter **??**.
- org.matsim.matrices.*, containing (somewhat ancient) helper classes to deal with matrices, in particular, origin-destination-matrices.
- org.matsim.population.*, mostly containing a collection of algorithms that go through the population and modify persons or plans.
- org.matsim.pt.*, see Chapter 16.
- org.matsim.run, see Section 44.3.2.
- org.matsim.utils.* containing various utilities such as the much-used ObjectAttributes (see Section 45.4.1).
- org.matsim.vehicles.*, see Section 6.6.
- org.matsim.vis.*, containing helper classes to write MATSim information, in particular from the mobsim, to file. This has to a large extent been superseded by the Via visualization package (see Chapter **??**).
- org.matsim.visum.*, containing code to input data from VISUM.
- org.matsim.withinday.*, see Chapter **??**.
- tutorial.*, containing example code of how to use MATSim, referenced throughout this book.

44.3.2 Core

The core is part of the main distribution (see the previous Section 44.3.1) and contains material that is considered basic and indispensable, and resides in the packages

- org.matsim.api.*
- org.matsim.core.*
- org.matsim.run.*

The MATSim core is maintained by the MATSim Core Developers Group.

44.3.3 Contributions

The idea of the contributions part of the repository is to host community contributions. Historically, most contributors are from the MATSim team, but this is not a requirement.[3] The code is maintained by the corresponding contributor. Code in this section of the repository is considered more stable than code in playgrounds. The Java packages often have the root org.matsim.contrib.*, but this is not mandatory.

At the time of writing, there are the following contributions (= extensions which are in the "contrib" part of the repository), listed in alphabetical order:

[3] It is currently at GitHub under https://github.com/matsim-org/matsim-libs/tree/master/contribs.

- accessibility, presented in Chapter **??**.
- analysis, presented in Chapter **??**.
- cadytsIntegration, presented in Chapter **??**.
- common is not a true contrib, i.e., it does not provide additional functionality by itself. Instead, it is a place where code used by several contribs, which has not yet made it into the main distribution is located. It also contains some long-running integration tests that are run at each build (i.e., more often than those contained in the integration contrib described below).
- dvrp, presented in Chapter 23.
- emissions, presented in Chapter **??**.
- freight, presented in Chapter **??**.
- freightChainsFromTravelDiaries, presented in Chapter **??**.
- grips, presented in Chapter **??**.
- gtfs2matsimtransitschedule, presented in Chapter **??**.
- integration is not a true contrib, i.e., it does not provide additional functionality. Instead, it is a place where integration tests that should run daily or weekly (instead of as often as possible) can be committed.
- locationchoice, presented in Chapter **??**.
- matrixbasedptrouter, presented in Chapter **??**.
- matsim4urbansim, presented in Chapter **??**.
- minibus, presented in Chapter **??**.
- multimodal, presented in Chapter **??**.
- networkEditor, presented in Chapter **??**.
- otfvis, presented in Chapter **??**.
- parking, presented in Chapter **??**.
- roadpricing, presented in Chapter **??**.
- socnetgen, presented in Chapter **??**.
- socnetsim, presented in Chapter **??**.
- transEnergySim, presented in Chapter **??**.
- wagonSim, presented in Chapter **??**.

44.3.4 Playgrounds

Another element of the MATSim repository is the "playgrounds". These are meant as a service to programmers. They have grown historically from the fact that MATSim's object classes and in consequence the interfaces between them have evolved and grown over time, and thus a stable API was not available. Regular code-wide refactorings, along the lines discussed, e.g., by Fowler (2004), were thus the norm for many years.

At this point, the extension points described in Chapter 45 should be somewhat stable and development against them should be possible without major changes from release to release. Anybody who needs tighter integration with the project should still apply for a playground.

44.3.5 Contributions and Extensions

Congruent with the structure of this book, the MATSim code structure contains a core which allows to run basic MATSim using the config file, a population and a network. Packages going beyond this basic functionality are extensions, where three different kind of extensions exist:

- extensions in the main distribution,[4]
- extensions contributed by the MATSim community known as contributions, and
- any code written anywhere published or unpublished extending the MATSim core.

Extensions are listed at `http://matsim.org/extensions`.

44.3.6 Releases, Nightly Builds and Code HEAD

Releases, nightly builds and the code head can be obtained from `http://matsim.org/downloads`.

MATSim releases are published approximately annually. Usually, MATSim users and MATSim power users as defined above in Section 44.2 work with releases.

MATSim uses continuous integration and, thus, nightly builds are available without stability guarantee under `http://matsim.org/downloads/nightly`. MATSim API developers that depend on a very recent feature might use Nightly builds.

Both Maven releases and Maven snapshots are available, see `http://matsim.org/downloads` for details.

MATSim API developers or core developers often work on the code's HEAD version that can be checked out from GitHub.

Nightly builds and maven snapshots are only generated when the code compiles and passes the regression tests. They are, in consequence, somewhat "safer" than the direct download from the HEAD.

44.4 Drivers, Organization and Tools of Development

Important drivers of the MATSim development are the projects and dissertations of the MATSim team. New features are developed as an answer to requirements of these dissertations and projects, where projects range from purely scientific ones—often sponsored by SNF (Schweizerischer Nationalfonds) or DFG (Deutsche Forschungsgemeinschaft)—via projects for governmental entities and projects where science and industry contribute equally—e.g., CTI (Commission for Technology and Innovation) projects—to purely commercial projects, which are managed by Senozon AG in the majority of cases. A significant number of innovations are also introduced by the collaboration with external researchers.

Systematic code integration is mainly performed by the Berlin group and by Senozon AG and Simunto GmbH. This includes continuous code review and integration upon request of the community, but also comprehensive code refactorings to clean up code and to improve modularity. Refactorings are discussed and documented in the MATSim issue tracker (`http://matsim.org/issuetracker`).

The development process is supported by a MATSim standing committee discussing software and sometimes conceptual issues on a regular basis (`http://matsim.org/committee`). Another element that brings in innovation as well as organization are the annual meetings. Right from the beginning, there have

[4] At the time of writing it is unclear if these extensions might one day become contributions, shrinking the MATSim main distribution to its core.

been a MATSim developer meetings focused on coding issues. Later, a user meeting offering insights into current work by the community has been added, sometimes combined with a tutorial. Finally, a conceptual meeting is now held every year, concentrating on issues that go beyond pure software engineering. The developer meeting and the conceptual meeting together establish the road map that guides development for the remainder of the year.

MATSim development makes use of a large number of tools, hopefully leading to better software quality. Historically, many of those tools ran from automated scripts and were made available at `http://matsim.org/developer`. Nowadays, most of them are automatically available from the build server (see `http://matsim.org/buildserver`) and/or from the repository (`https://github.com/matsim-org/matsim-libs`), so that many of them are scheduled for removal from `http://matsim.org/developer`. Some of these tools are: a change log; an issue tracker; the javadoc documentation; static code analyses performed by *FindBugs* and *PMD*; test code coverage analysis; copy paste analysis; code metrics; Maven dependencies; and information about the nightly test results. These nightly test results are generated by the MATSim build server based on the Jenkins software.

Furthermore, there is a MATSim benchmark at `http://matsim.org/files/benchmark/benchmark.zip`. For results see `http://matsim.org/benchmark`.

Most MATSim developers use Eclipse as an IDE. The MATSim documentation is tailored to this IDE. Team development is currently based on Git as revision control system. External library dependencies are managed by Maven.

44.5 Documentation, Dissemination and Support

The main documentation is now this book. Additional information, including tutorials, can be found under `http://matsim.org/docs`. Code documentation in form of javadoc can be found unter `http://matsim.org/javadoc`.

For fast application of MATSim, some small-scale example scenarios are provided in the code base (`folder: examples`), where recently an extended version of the well-known benchmark scenario for the City of Sioux Falls has been added (Chakirov and Fourie, 2014) (Chapter **??**). Additional example datasets, including Berlin datasets, can be obtained via `http://matsim.org/datasets`.

Further information is disseminated at the afore-described annual user meetings and MATSim mailing lists, see `http://matsim.org/mailinglists`. Support is provided by the MATSim team via these mailing lists and via `http://matsim.org/faq`, both on a best effort basis. Many components of MATSim are documented by the numerous papers published in international journals and presented at worldwide conferences. Information about such publications can, e.g., be obtained from `http://matsim.org/publications` and from this book.

44.6 Your Contribution to MATSim

The technical details, i.e., the MATSim extension points, on where to hook with MATSim are detailed in Chapter 45. Here, the different ways of contributing to MATSim according to the roles presented in Section 44.2 are introduced.

As a MATSim user, power user, or API developer, you are warmly welcome to make an important impact by reporting your achievements, needs and problems with, or bugs of, the software via the users mailing list, the issue tracker, the FAQ, or at the annual MATSim user meeting.

If you would like to directly contribute to the code base of MATSim, you are welcome to become part of the contributions repository.

If you are the type of person that likes to change the core system, you can, although it is a long way, become a member of the MATSim core developers group. Core developers are usually picked from the MATSim team. Prerequisites are a strong computer scientist background, several years of experience with MATSim and a deep understanding of large software projects.

Agent-Based Traffic Assignment

Choice Models in MATSim

Queueing Representation of Kinematic Waves

Research Avenues

Choice Set Generation

Acronyms

AI Artificial Intelligence. 42

API Application Programming Interface. 9, 25, 93, 102, 108, 110–112, 119

CAS Complex Adaptive Systems. 3

CTI Commission for Technology and Innovation. 111

CUDA Compute Unified Device Architecture, a parallel computing platform and API by NVIDIA. 102

DEQSim Discrete Event Queue Simulation. 6, 7, 36

DFG Deutsche Forschungsgemeinschaft. 111

DRT Demand Responsive Transport. 91

DTD Document Type Description. 61, 67

DVRP Dynamic Vehicle Routing Problem. 84–86, 89–91

EMME Equilibre Multimodal Multimodal Equilibrium. See `http://www.inrosoftware.com/en/products/emme/`. 26, 62

EPSG European Petroleum Survey Group. 19

ETH Eidgenössische Technische Hochschule. 105

FCL Future Cities Laboratory. 105

FIFO First In, First Out. 73, 90

GIS Geographic Information System. 19

GPLv2 GNU General Public License version 2.0. 106

GPS Global Positioning System. 19, 20

GTFS General Transit Feed Specification. 81

GUI Graphical User Interface. 10, 11, 22, 24

HAFAS HaCon Fahrplan-Auskunfts-System. 81

HPCC High-Performance Computing Clusters. 34

ICT Information and Communications Technology. 83

IDE Integrated Development Environment. 10, 25, 112

ILS Institut für Land- und Seeverkehr – Institute for Land and Sea Transport Systems. 105

IVT Institut für Verkehrsplanung und Transportsysteme – Institute for Transport Planning and Systems. 105

JAR Java ARchive. 10

Java SE Java Standard Edition. 9

JAXB Java Architecture for XML Binding. 109

JDEQSim Java Discrete Event Queue Simulation. 6, 7, 34, 36

JOSM Java Open Street Map Editor. 23

MATSim Multi-Agent Transport Simulation. See `http://www.matsim.org`. 3–7, 9–12, 14–20, 22–28, 31–36, 38, 41, 42, 45, 47, 49–51, 53, 58–62, 65, 70, 71, 73, 75–77, 80–82, 84, 86–89, 93, 95–100, 102–106, 108–113, 123

MNL Multinomial Logit Model. 39

mobsim Mobility simulation. Also, depending on the context and specific mobility simulation capabilities, called network loading, traffic flow simulation or synthetic reality. 5, 6, 11, 12, 14, 16, 17, 22, 33–35, 47, 50, 93–98, 101–103, 109, 124

MSA Method of Successive Averages. See, e.g., Liu et al. (2007). 39, 50, 51

O-D Origin-Destination. 101

OSM OpenStreetMap (OpenStreetMap, 2015). 20, 60, 62, 124

OTFVis On The Fly Visualizer. 10, 14

PCU Passenger Car Unit. 80

PhD Philosophiae Doctor – Doctor of Philosophy. 106

PT Public Transport. 77, 80, 81

QGIS Quantum GIS. 19, 26

Scala SCAlable LAnguage. See `http://www.scala-lang.org/`. 26

SI Système International (d'Unités): International System (of Units). 57

SNF Schweizerischer Nationalfonds. 111

SPI Service Provider Interface. 95

SUMO Simulation of Urban Mobility. See `http://www.dlr.de/ts/sumo/en/`. 26

TRANSIMS TRansportation ANalysis and SIMulation System. See `https://code.google.com/archive/p/transims/`. 3

TU Technische Universität. 26, 105

URL Uniform Resource Locator. 12, 22

UTM Universal Transverse Mercator. 18, 19

VISUM Verkehr In Städten – UMlegung. See http://www.ptv.de. 26, 62, 76, 81, 109

VPL VerkehrsPLanung. See http://www.ivt.ethz.ch/vpl/. 105

VRP Vehicle Routing Problem. 84, 85

VSP VerkehrsSystemPlanung und Verkehrstelematik – The Transport Systems Planning and Transport Telematics group at TU Berlin. See https://www.vsp.tu-berlin.de. 105

VTTS Value of Travel Time Savings. 46, 48

WKT Well-Known Text. 19

XML Extensible Markup Language, see http://www.w3.org/XML/. 14, 59, 61, 95, 104, 109

Glossary

MATSim example project
The github project at `https://github.com/matsim-org/matsim-example-project` that is meant to be forked/cloned, and used as a starting point for using or extending MATSim.. 104

Activity The central element of modern activity-based modeling (see below). 4, 125

Activity location People perform activities at activity locations, which can be as small as one single building or large zones. In MATSim, activity locations are often further specified by using the facility object, which (in addition to others) define open times. 124

Activity-based Modern transport planning assumes that "*travel demand is derived from activity demand*" (Jones, 1979; Bowman, 2009a,b; Bhat and Koppelman, 2003; Ettema and Timmermans, 1997; Bowman and Ben-Akiva, 1996, 2001). People travel because they want to perform a certain activity, which is best captured by activity-based models with activities the central element of modeling. 4

Algorithm A set of operations to solve a specific problem. 7, 49

C++ An object-oriented programming language with full control of memory management. 6

command line A method to give input to computers.. 11, 79

Configuration file The main configuration screw for MATSim, often just referred to as `config file` or as `config.xml`. Also see config. 10–14, 17, 19, 22, 24, 31, 33–40, 57, 58, 60, 73, 80, 104, 108, 111, 123

Configuration object The object in the MATSim code containing configuration options. It can be modified by the config file, but also by other mains, in particular by scripts-in-Java. 20, 31, 34, 37, 94, 99, 123

Contribution An extension contributed by the MATSim community and hosted in the MATSim repository. See `http://matsim.org/extensions`.. 25, 84, 109, 111, 112

Eclipse The standard integrated development environment (IDE) used by the MATSim developers. 10, 112

Equilibrium A system state where are competing forces are balanced. 7, 8

Event Small pieces of information reported by the mobsim, describing a simulation object action at a specific time. 17, 94, 96, 97, 101–103

Extension Core MATSim uses only a config file, population file and network file, corresponding to the book's part I. An extension is any code that extends this core MATSim, corresponding to this

book's part II. They hook to MATSim via the extension points described in Chapter 45. 25, 37, 95, 104, 108, 111, 123

Facility An optional element in MATSim to further specify an activity location. 59, 123

Framework A software concept, providing generic functionality and application-specific software. It is selectively changed by user code. MATSim is currently a framework, but is developing towards also being useful as a library/toolbox. 4, 101, 102

Git A free and open source distributed version control system. 108, 112

GitHub A web-based Git repository hosting service, see `https://github.com/`. 9, 10, 108, 109, 111

Google Earth Google's virtual globe. 19

Identifier A name that labels an object in a unique way. 19, 69, 71, 95, 96

Iteration Numerical equilibrium search methods, such as MATSim, are iterative. A MATSim run is thus composed of a configurable number of iterations. 11, 17, 124

Java A modern, object-oriented, cross-platform programming language run in virtual machines. 4, 10, 24–26, 34, 36, 61, 72, 80, 95, 96, 99, 102, 104, 108, 109, 123, 124

Javadoc Source code documentation compiled from javadoc annotations in the source files. 112

Jenkins A software tool for continuous integration. 112

Large-scale Denoting large, extended simulation scenarios, often modeling complete cities, or even countries. 3

Leg A plan element, part of a trip performed with a specific mode. In transport planning, this is often called a stage. 16, 101, 102, 125

Link A network component representing streets. 14

MATSim run A configurable number set of iterations, typically ending with an equilibrium solution of transport supply and demand. 4, 10, 17, 33, 104, 124

Maven A build automation tool by Apache tailored to Java. 10, 25, 93, 104, 108, 111, 112

Module According to Merriam-Webster (`http://www.merriam-webster.com`), a module is "one of a set of parts that can be connected or combined to build or complete something" or more specifically "a part of a computer or computer program that does a particular job". That is, "module" is not a very specific term, and, in consequence, modules exist in MATSim at many levels. 59

Multimodal Combining different means of transport. 26, 35, 73, 75

Node An element of a MATSim network representing intersections. Note that intersections are not modeled explicitly in MATSim, i.e., cars do not interact at intersections. 14

Objective function A central element in optimization problems, among others. An objective function, sometimes also called loss or cost function, is mapping of candidate solutions onto a real number. 125

Osmosis Command line Java application for processing OSM data. See `http://wiki.openstreetmap.org/wiki/Osmosis`. 62

Plan The agent's day schedule and, after run completion, an associated score. 4, 16, 68, 97

QSim The standard MATSim mobsim. 6, 7, 14, 33–35, 65, 68–73, 76, 94

Replanning The stage when agents modify their plans. 5, 12, 16, 36, 93, 97, 103

Scenario In MATSim context, a scenario is defined as: the combination of specific agent populations, their initial plans and activity locations (home, work, education), the network and facilities where, and on which, they compete in time-space for their slots and modules, i.e., behavioral dimensions, which they can adjust during their search for equilibrium. 6, 9, 12, 94, 102

Score After execution in the infrastructure, the agents' day plans are evaluated through an individual objective function, the MATSim scoring function. Also see utility. 4, 42, 103, 124

Scoring see score. 36, 103

Senozon AG A spin-off company founded by two core developers of MATSim. 81, 106, 111

Simunto GmbH A company founded by a core developer of MATSim. 106, 107, 111, 125

Stage A stage is part of a trip, performed with a single mode. In MATSim called leg. 124

Study The basic organizational unit of research in empirical science. Comparable to the experiment in natural sciences. 4

Teleportation Moving vehicles from origin to destination, at a predefined speed, without considering interactions in the network. 16, 68, 71, 73, 75

Teleported see teleportation. 75

Trip The connection between two activities, composed of multiple legs. 18, 124

Utility A central economic concept representing satisfaction through goods consumption. The MATSim score can be interpreted in utility units. 42, 125

Via Visualization and analysis tool for MATSim by Simunto GmbH. 10, 14, 23, 109

Bibliography

Agarwal, A., M. Zilske, K. Rao and K. Nagel (2015) An elegant and computationally efficient approach for heterogeneous traffic modelling using agent based simulation, *Procedia Computer Science*, **52** (C) 962–967, doi:10.1016/j.procs.2015.05.173.

Arnott, R., A. de Palma and R. Lindsey (1990) Economics of a bottleneck, *Journal of Urban Economics*, **27** (1) 111–130, doi:10.1016/0094-1190(90)90028-L.

Arnott, R., A. de Palma and R. Lindsey (1993) A structural model of peak-period congestion: A traffic bottleneck with elastic demand, *The American Economic Review*, **83** (1) 161–179.

ASTRA (2006) Swiss federal roads authority, webpage, http://www.astra.admin.ch/.

Balmer, M., A. Horni, K. Meister, F. Ciari, D. Charypar and K. W. Axhausen (2009) Wirkungen der Westumfahrung Zürich: Eine Analyse mit einer Agenten-basierten Mikrosimulation, *Final Report*, Baudirektion Kanton Zurich, IVT, ETH Zurich, Zurich, February 2009.

Balmer, M., K. Meister, R. A. Waraich, A. Horni, F. Ciari and K. W. Axhausen (2010) Agenten-basierte Simulation für location based services, *Final Report*, F&E Förderung: Science to Market, **KTI 8443.1 ESPP-ES**, Datapuls AG, IVT, ETH Zurich, Zurich, February 2010.

Balmer, M., B. Raney and K. Nagel (2005) Adjustment of activity timing and duration in an agent-based traffic flow simulation, in: H. Timmermans (ed.) *Progress in activity-based analysis*, 91–114, Elsevier, Oxford.

Barcelo, J., H. Grzybowska and S. Pardo (2007) Vehicle routing and scheduling models, simulation and city logistics, 163–195, Springer, NY.

Ben-Akiva, M. E. and S. R. Lerman (1985) *Discrete Choice Analysis: Theory and Application to Travel Demand*, MIT Press, Cambridge.

Bhat, C. R. and F. S. Koppelman (2003) Activity-based modeling of travel demand, in: R. W. Hall (ed.) *Handbook of Transportation Science*, 39–65, Springer, New York.

Bowman, J. L. (2009a) Historical development of activity based model theory and practice (part 1), *Traffic Engineering and Control*, **50** (2) 59–62.

Bowman, J. L. (2009b) Historical development of activity based model theory and practice (part 2), *Traffic Engineering and Control*, **50** (7) 314–318.

Bowman, J. L. and M. E. Ben-Akiva (1996) Activity-based travel forecasting, in: *Activity-Based Travel Forecasting Conference*, New Orleans, June 1996.

Bowman, J. L. and M. E. Ben-Akiva (2001) Activity-based disaggregate travel demand model system with activity schedules, *Transportation Research Part A: Policy and Practice*, **35** (1) 1–28.

Certicky, M., M. Jakob, R. Pibil and Z. Moler (2014) Agent-based simulation testbed for on-demand transport services, in: *2014 International Conference on Autonomous Agents and Multi-agent Systems*, AAMAS '14, 1671–1672, Richland, SC.

Cetin, N. (2005) Large-scale parallel graph-based simulations, Ph.D. Thesis, ETH Zurich, Zurich.

Cetin, N., A. Burri and K. Nagel (2003) A large-scale agent-based traffic microsimulation based on queue model, in: *Swiss Transport Research Conference (STRC)*, Monte Verita. See http://www.strc.ch.

Chakirov, A. and P. J. Fourie (2014) Enriched Sioux Falls Scenario with Dynamic and Disaggregate Demand, *Working Paper*, Future Cities Laboratory, Singapore-ETH Centre (SEC), Singapore.

Charypar, D. (2008) Efficient algorithms for the microsimualtion of travel behavior in very large scenarios, Ph.D. Thesis, ETH Zurich, Zurich.

Charypar, D., K. Axhausen and K. Nagel (2007a) An event-driven parallel queue-based microsimulation for large scale traffic scenarios, in: *World Conference on Transport Research (WCTR'07)*, Berkeley, CA. Also VSP WP 07-03.

Charypar, D., K. Axhausen and K. Nagel (2007b) Event-driven queue-based traffic flow microsimulation, *Transportation Research Record*, **2003**, 35–40, doi:10.3141/2003-05.

Charypar, D., M. Balmer and K. W. Axhausen (2009) High-performance traffic flow microsimulation for large problems, in: *88th Annual Meeting of the Transportation Research Board*, Washington, D.C., January 2009.

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

Ciari, F., M. Balmer and K. W. Axhausen (2007) Mobility tool ownership and mode choice decision processes in multi-agent transportation simulation, in: *7th Swiss Transport Research Conference*, Ascona, September 2007.

Ciari, F., M. Balmer and K. W. Axhausen (2008) A new mode choice model for a multi-agent transport simulation, in: *8th Swiss Transport Research Conference*, Ascona, October 2008.

Dijkstra, E. W. (1959) A note on two problems in connexion with graphs, *Numerische Mathematik*, **1**, 269–271.

Dobler, C. (2009) Implementations of within day replanning in MATSim-T, *Working Paper*, **598**, IVT, ETH Zurich, Zurich.

Dobler, C. (2010) Implementation of a time step based parallel queue simulation in MATSim, in: *10th Swiss Transport Research Conference*, Ascona, September 2010.

Dobler, C. (2013) Travel behaviour modelling for scenarios with exceptional events - methods and implementations, Ph.D. Thesis, ETH Zurich, Zurich.

Dobler, C. and K. W. Axhausen (2011) Design and implementation of a parallel queue-based traffic flow simulation, *Working Paper*, **732**, IVT, ETH Zurich, Zurich.

Eiben, A. E. and J. E. Smith (eds.) (2003) *Introduction to Evolutionary Computing*, Springer, Berlin.

Ettema, D. F. and H. J. P. Timmermans (eds.) (1997) *Activity-Based Approaches to Travel Analysis*, Pergamon, Oxford.

FHWA (2013) TRANSIMS Background, webpage, http://www.fhwa.dot.gov/planning/tmip/resources/transims/.

Fowler, M. (2004) *Refactoring: Improving the design of existing code*, Addison-Wesley.

Frejinger, E. and M. Bierlaire (2007) Capturing correlation with subnetworks in route choice models, *Transportation Research Part B: Methodological*, **41** (3) 363–378.

Gawron, C. (1998) An iterative algorithm to determine the dynamic user equilibrium in a traffic simulation model, *International Journal of Modern Physics C*, **9** (3) 393–408.

Geotools (accessed 2015) The open source Java GIS toolkit, webpage. http://www.geotools.org.

GitHub (2015) Web-based git repository hosting service, webpage, https://github.com.

Grether, D., Y. Chen, M. Rieser and K. Nagel (2009) Effects of a simple mode choice model in a large-scale agent-based transport simulation, in: A. Reggiani and P. Nijkamp (eds.) *Complexity and Spatial Networks. In Search of Simplicity*, Advances in Spatial Science, 167–186, Springer, doi:10.1007/978-3-642-01554-0.

Grether, D., A. Neumann and K. Nagel (2011) Traffic light control in multi-agent transport simulations, *VSP Working Paper*, **11-08**, TU Berlin, Transport Systems Planning and Transport Telematics, Berlin. URL.

Grether, D., A. Neumann and K. Nagel (2012) Simulation of urban traffic control: A queue model approach, *Procedia Computer Science*, **10**, 808–814, doi:10.1016/j.procs.2012.06.104.

Guo, J. Y. and C. R. Bhat (2007) Population synthesis for microsimulating travel behavior, *Transportation Research Record*, **2014** (12) 92–101.

Horni, A. and K. W. Axhausen (2012) MATSim agent heterogeneity and week scenario, *Working Paper*, **836**, IVT, ETH Zurich, Zurich.

Horni, A. and D. Grether (2007) Counts, internal presentation, MATSim-T Workshop, IVT, ETH Zurich, Castasegna, October 2007.

Horni, A., K. Nagel and K. W. Axhausen (2012) High-resolution destination choice in agent-based models, *Annual Meeting Preprint*, **12-1988**, Transportation Research Board, Washington, D.C. Also VSP WP 11-17.

Horni, A., D. M. Scott, M. Balmer and K. W. Axhausen (2009) Location choice modeling for shopping and leisure activities with MATSim: Combining micro-simulation and time geography, *Transportation Research Record*, **2135**, 87–95.

Horni, A., B. J. Vitins and K. W. Axhausen (2011) The Zurich scenario: A technical overview, *Working Paper*, **687**, IVT, ETH Zurich, Zurich.

Jacob, R. R., M. V. Marathe and K. Nagel (1999) A computational study of routing algorithms for realistic transportation networks, *ACM Journal of Experimental Algorithms*, **4** (1999es, Article No. 6), doi:10.1145/347792.347814.

Jones, P. M. (1979) New approaches to understand travel behaviour: the human activity approach, in: D. A. Hensher and P. R. Stopher (eds.) *Behavioural Travel Modelling*, 55–80, Croom Helm Ltd, Kent.

Lämmel, G., D. Grether and K. Nagel (2010) The representation and implementation of time-dependent inundation in large-scale microscopic evacuation simulations, *Transportation Research Part C: Emerging Technologies*, **18** (1) 84–98, doi:10.1016/j.trc.2009.04.020.

Lefebvre, N. and M. Balmer (2007) Fast shortest path computation in time-dependent traffic networks, presentation, The 7th Swiss Transport Research Conference (STRC), Ascona, September.

Liao, T.-Y., T.-Y. Hu and D.-J. Chen (2008) Object-oriented evaluation framework for dynamic vehicle routing problems under real-time information, *Annual Meeting Preprint*, **08-2222**, Transportation Research Board, Washington, D.C.

Liu, H., X. He and B. He (2007) Method of successive weighted averages (MSWA) and self-regulated averaging schemes for solving stochastic user equilibrium problem, *Networks and Spatial Economics*, **9** (4) 485–503.

Maciejewski, M. (2014) Online taxi dispatching via exact offline optimization, *Logistyka*, **3**, 2133–2142.

Maciejewski, M. and K. Nagel (2012) Towards multi-agent simulation of the dynamic vehicle routing problem in MATSim, in: R. Wyrzykowski et al (ed.) *Parallel Processing and Applied Mathematics (PPAM), Revised Selected Papers, Part II*, Lecture Notes in Computer Science, Springer, Berlin, doi:10.1007/978-3-642-31500-8_57.

Maciejewski, M. and K. Nagel (2013a) A microscopic simulation approach for optimization of taxi services, in: T. Albrecht, B. Jaekel and M. Lehnert (eds.) *3rd International Conference on Models and Technologies for Intelligent Transportation Systems 2013*, 1–10, TUDpress. Also VSP WP 13-12.

Maciejewski, M. and K. Nagel (2013b) Simulation and dynamic optimization of taxi services in MATSim, *VSP Working Paper*, **13-05**, TU Berlin, Transport Systems Planning and Transport Telematics. URL.

Maciejewski, M. and K. Nagel (2014) The influence of multi-agent cooperation on the efficiency of taxi dispatching, in: *10th International Conference Parallel Processing and Applied Mathematics (PPAM) Part II*, no. 8385 in: LNCS, Warsaw, Poland, doi:10.1007/978-3-642-55195-6_71.

MATSim (2016) Multi-Agent Transportation Simulation, webpage, http://www.matsim.org.

Meister, K. (2008) Erstellung von MATSim Facilities für das Schweiz-Szenario, *Working Paper*, **541**, IVT, ETH Zurich.

Meister, K. (2011) Contribution to agent-based demand optimization in a multi-agent transport simulation, Ph.D. Thesis, ETH Zurich, Zurich.

Meister, K., M. Balmer, F. Ciari, A. Horni, M. Rieser, R. A. Waraich and K. W. Axhausen (2010) Large-scale agent-based travel demand optimization applied to Switzerland, including mode choice, in: *12th World Conference on Transportation Research*, Lisbon, July 2010.

Müller, K. (2011) IPF within multiple domains: Generating a synthetic population for Switzerland, in: *11th Swiss Transport Research Conference*, Ascona, May 2011.

Neumann, A. (2008) Modellierung und Evaluation von Lichtsignalanlagen in Queue-Simulationen, Diplomarbeit (Diploma Thesis), TU Berlin, Institute for Land and Sea Transport Systems, Berlin. Also VSP WP 08-24.

OpenStreetMap (2015) The Free Wiki World Map, webpage, `http://www.openstreetmap.org`.

Pawlak, J., A. Sivakumar and J. W. Polak (2011) The consequences of the productive use of travel time: Revisiting the goods-leisure trade-off in the era of pervasive ICT, in: *2nd International Choice Modelling Conference*, Leeds, July 2011.

Popovici, E., R. P. Wiegand and E. D. De Jong (2012) Coevolutionary principles, in: G. Rozenberg, T. Bäck and J. N. Kok (eds.) *Handbook of Natural Computing*, 987–1033, Springer, Heidelberg.

PTV (2011) *VISUM 12 - New features at a glance*, PTV, Karlsruhe.

Raney, B. (2005) Learning framework for large-scale multi-agent simulations, Ph.D. Thesis, ETH Zurich, Zurich.

Raney, B. and K. Nagel (2006) An improved framework for large-scale multi-agent simulations of travel behaviour, in: P. Rietveld, B. Jourquin and K. Westin (eds.) *Towards better performing European Transportation Systems*, 305–347, Routledge, London.

Redmond, L. S. and G. Mokhtarian (2001) The positive utility of the commute: Modeling ideal commute time and relative desired commute amount, *Transportation*, **28** (2) 179–205.

Regan, A., H. Mahmassani and P. Jaillet (1998) Evaluation of dynamic fleet management systems: Simulation framework, *Transportation Research Record*, **1645**, 176–184.

Reinhold, T. (2006) Konzept zur integrierten Optimierung des Berliner Nahverkehrs, in: *Öffentlicher Personennahverkehr*, 131–146, Springer, Berlin, doi:10.1007/3-540-34209-5_8.

Rieser, M. (2010) Adding transit to an agent-based transportation simulation: Concepts and implementation, Ph.D. Thesis, TU Berlin, Berlin, doi:10.14279/depositonce-2581.

Rieser, M., D. Grether and K. Nagel (2009) Adding mode choice to a multi-agent transport simulation, *Transportation Research Record*, **2132**, 50–58, doi:10.3141/2132-06.

Ronald, N., R. G. Thompson and S. Winter (2014) Simulating demand-responsive transportation: A review of agent-based approaches, *submitted to Transport Reviews*.

Ronald, N., R. G. Thompson and S. Winter (2015) A comparison of constrained and ad-hoc demand-responsive transportation systems, in: *94th Transportation Research Board Annual Meeting, Washington, D.C., January 2015*.

Russel, S. J. and P. Norvig (2010) *Artificial Intelligence – A Modern Approach*, Pearson Education, Upper Saddle River.

Simon, P. M., J. Esser and K. Nagel (1999) Simple queueing model applied to the city of Portland, *International Journal of Modern Physics C*, **10** (5) 941–960.

Smith, L., R. Beckman, D. Anson, K. Nagel and M. Williams (1995) TRANSIMS: TRansportation ANalysis and SIMulation System, in: *5th National Transportation Planning Methods Applications Conference*, Seattle, WA.

Strippgen, D. (2009) Investigating the technical possibilities of real-time interaction with simulations of mobile intelligent particles, Ph.D. Thesis, TU Berlin, Berlin.

Swiss Federal Statistical Office (BFS) (2000) Eidgenössische Volkszählung 2000, `http://www.bfs.admin.ch/bfs/portal/de/index/infothek/erhebungen__quellen/blank/blank/vz/uebersicht.html`.

Swiss Federal Statistical Office (BFS) (2001) *Eidgenössische Betriebszählung 2001 – Sektoren 2 und 3, GEOSTAT Datenbeschreibung*, Swiss Federal Statistical Office (BFS), GEOSTAT, Neuchatel, http://www.bfs.admin.ch/bfs/portal/de/index/dienstleistungen/geostat/datenbeschreibung/betriebszaehlung05.Document.111424.pdf.

Swiss Federal Statistical Office (BFS) (2006) *Ergebnisse des Mikrozensus 2005 zum Verkehrsverhalten*, Swiss Federal Statistical Office (BFS), Neuchatel.

Train, K. E. (2003) *Discrete Choice Methods with Simulation*, Cambridge University Press, New York.

TRANSIMS Open Source (2013) Getting Started with TRANSIMS, webpage, http://code.google.com/p/transims/wiki/GettingStarted.

Vickrey, W. S. (1969) Congestion theory and transport investment, *The American Economic Review*, **59** (2) 251–260.

Waraich, R. A., D. Charypar, M. Balmer and K. W. Axhausen (2009) Performance improvements for large scale traffic simulation in MATSim, in: *9th Swiss Transport Research Conference*, Ascona, September 2009.

Zilske, M. and K. Nagel (2015) A simulation-based approach for constructing all-day travel chains from mobile phone data, *Procedia Computer Science*, **52**, 468–475, doi:10.1016/j.procs.2015.05.017.