

# 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: April 25, 2024—



# Contents

Title Page	1
Table of Contents	i
<b>I Using MATSim</b>	<b>1</b>
<b>1 Introducing MATSim</b>	<b>3</b>
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
<b>2 Let’s Get Started</b>	<b>9</b>
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 . . . . .	13
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 Data Requirements . . . . .	18
2.3.4 Units, Conventions, and Coordinate Systems . . . . .	19
2.3.5 Open Scenario Input Data . . . . .	21
2.4 MATSim Survival Guide . . . . .	22
<b>3 Available Functionality and How to Use It</b>	<b>25</b>
3.1 MATSim Modularity . . . . .	25
3.2 Levels of Access . . . . .	26
3.2.1 Using MATSim through the GUI with config, network, and population only . . . . .	26
3.2.2 Using MATSim through the GUI with additional files . . . . .	26
3.2.3 Writing “Scripts in Java” . . . . .	26
3.2.4 Using Contribs or External Extensions . . . . .	27
3.2.5 Writing Your Own Extensions . . . . .	27
3.3 The Ideas Behind this Setup . . . . .	28
3.4 An Overview of Existing MATSim Functionality . . . . .	29
<b>4 More About Configuring MATSim</b>	<b>33</b>
4.1 General . . . . .	33
4.2 MATSim Data Containers . . . . .	33

4.2.1	Network	33
4.2.2	Population	33
4.2.3	Further MATSim containers	34
4.3	Global Modules and Global Aspects	34
4.3.1	controler config file section	34
4.3.2	Parallel Computing	34
4.3.3	global config file section	35
4.4	Mobility Simulations	35
4.4.1	QSim	36
4.4.2	JDEQSim	37
4.4.3	Hermes	37
4.5	Scoring	37
4.6	Replanning Strategies	38
4.6.1	Plans Generation and Removal (Choice Set Generation)	38
4.6.2	Plan Selection (Choice)	40
4.6.3	Innovation Switch-Off	41
4.7	Observational Modules	41
4.7.1	Travel Time Calculator	41
4.7.2	Link Stats	41
4.8	More Information	41
<b>II</b>	<b>Visualizers</b>	<b>43</b>
<b>5</b>	<b>Simunto Via</b>	<b>44</b>
5.1	Basic Information	44
5.2	Introduction	44
5.3	Simple Usage	45
5.4	Use Cases and Examples	47
5.4.1	Agent Visualization	47
5.4.2	Facility Analysis	47
5.4.3	Public Transport Analysis	48
5.4.4	Scenario Comparisons	49
5.4.5	Aggregating Data	49
<b>6</b>	<b>SimWrapper</b>	<b>51</b>
6.1	Basic Information	51
6.2	Introduction	52
6.3	Starting SimWrapper	52
6.4	Using SimWrapper to view MATSim results	53
6.4.1	SimWrapper views based on regular MATSim output	53
6.4.2	SimWrapper dashboard generated by the SimWrapper contrib	53
6.4.3	Self-configured SimWrapper dashboards	53
6.5	Building interactive dashboards with SimWrapper	54
6.6	To learn more	55
<b>7</b>	<b>OTFVis</b>	<b>57</b>
7.1	Basic Information	57
7.2	Introduction	57
7.3	Quickstart	58
7.4	More Ways of Using OTFVis	58
7.4.1	MVI Files (unmaintained)	58
7.4.2	Starting OTFVis from the commandline (unmaintained)	58
7.4.3	Use Cases of OTFVis (unmaintained)	59
7.4.4	Viewing an MVI File (unmaintained)	60
7.4.5	General Interaction with the Main Screen	61

7.4.6	User Interaction in the Live Mobsim	61
7.4.7	Running a Query in OTFVis Real Time Data	62
7.5	Extending OTFVis	63
7.5.1	Design Principles of OTFVis	63
7.5.2	Readers and Writers	64
7.5.3	Visualization of the Data	64
7.5.4	Layers	64
7.5.5	Patching the Connections	65
7.5.6	Sending the Data	65
7.5.7	Performance Considerations	65
7.5.8	Sending Live Data	65
<b>III</b>	<b>Generation of MATSim input</b>	<b>67</b>
<b>8</b>	<b>Generation of the Initial MATSim Input</b>	<b>68</b>
8.1	Coordinate Transformations in Java	68
8.2	Network Generation	69
8.2.1	From OpenStreetMap	69
8.2.2	From Other Sources	69
8.3	Initial Demand Generation	69
8.3.1	Simple Initial Demand	69
8.3.2	Realistic Initial Demand	70
<b>9</b>	<b>Eqasim</b>	<b>71</b>
9.1	Basic Information	71
<b>IV</b>	<b>Additional core features</b>	<b>73</b>
<b>10</b>	<b>Additional MATSim Data Containers</b>	<b>74</b>
10.1	Attributable	74
10.2	Time-Dependent Network	75
10.3	Subpopulations	75
10.4	Counts	76
10.5	Facilities	77
10.6	Households	78
10.7	Vehicles	78
<b>11</b>	<b>Multi-modal QSim and its interaction with routing</b>	<b>79</b>
11.1	Other Modes than Car: Introduction	79
11.2	Other Modes than Car: Routing Side	80
11.2.1	Routing Side: Teleportation	80
11.2.2	Routing Side: Network Modes	81
11.2.3	Routing Side: Passenger Modes	82
11.2.4	Other Routing Options	82
11.3	Other Modes than Car: QSim Side	82
11.3.1	QSim Side: Teleportation	82
11.3.2	QSim Side: Network Modes	83
11.3.3	QSim Side: Explicitly Simulated Passenger Modes	83
11.3.4	QSim Side: Departure Handlers	84
11.4	Other Modes than Car: Scoring Side	85
11.5	Vehicle Types and Vehicles	85
11.5.1	Vehicle sources and vehicle IDs	85
11.5.2	Vehicle Placement and Behavior	86
11.6	Legs, trips, and interaction activities	87

11.7	Access/egress routing . . . . .	88
11.8	Other . . . . .	88
<b>12</b>	<b>Modeling Public Transport with MATSim</b>	<b>89</b>
12.1	Basic Information . . . . .	89
12.2	Introduction . . . . .	89
12.3	Data Model and Simulation Features . . . . .	90
12.4	File formats . . . . .	91
12.4.1	transitVehicles.xml . . . . .	91
12.4.2	transitSchedule.xml . . . . .	91
12.4.3	Events for a public transit trip . . . . .	93
12.5	Swiss Rail Raptor . . . . .	94
12.6	Running with small sample sizes . . . . .	94
12.7	Possible Improvements . . . . .	94
<b>10</b>	<b>A Closer Look at Scoring</b>	<b>97</b>
10.1	Good Plans and Bad Plans, Score and Utility . . . . .	97
10.2	The Current Charypar-Nagel Utility Function . . . . .	98
10.2.1	Mathematical Form . . . . .	98
10.2.2	Basic Function . . . . .	98
10.2.3	Activities . . . . .	99
10.2.4	Travel . . . . .	101
10.2.5	Illustration . . . . .	102
10.2.6	The “Wrapping Around” of the Utility Function . . . . .	103
10.2.7	MATSim Scoring, Opportunity Cost of Time, and the VTTS . . . . .	103
10.2.8	The Resulting Modeling of Schedule Delay Costs . . . . .	104
10.3	Implementation Details . . . . .	105
10.3.1	Negative Durations . . . . .	105
10.3.2	Score Averaging . . . . .	106
10.3.3	Forcing Scores to Converge . . . . .	106
10.4	Typical Scoring Function Parameters and their Calibration . . . . .	107
10.5	Helper functions . . . . .	108
10.5.1	Experienced plans . . . . .	108
10.6	Applications and Extensions of the Scoring Function . . . . .	108
10.7	Appendix: Old Version of the Zero Utility Duration . . . . .	109
<b>V</b>	<b>Analysis</b>	<b>111</b>
<b>11</b>	<b>CSV outputs</b>	<b>112</b>
11.1	MATSim standard CSV outputs . . . . .	112
11.2	Recommendations for using CSV in the MATSim context . . . . .	112
11.2.1	CSV file format variations: delimiters, header, quotes, and comments . . . . .	113
11.2.2	Guidance on creating MATSim-standard CSV files . . . . .	113
<b>12</b>	<b>Python bindings</b>	<b>115</b>
12.1	Basic Information . . . . .	115
12.2	Introduction . . . . .	116
12.3	Installation . . . . .	116
12.4	Using the MATSim Python API . . . . .	116
<b>13</b>	<b>R bindings</b>	<b>119</b>
13.1	Basic Information . . . . .	119
13.2	Introduction . . . . .	119
13.3	Installation . . . . .	120
13.4	Using the R API . . . . .	120

<b>14</b>	<b>The “Analysis” Contribution</b>	<b>121</b>
14.1	Basic Information . . . . .	121
14.2	Summary . . . . .	121
<b>15</b>	<b>Emissions</b>	<b>123</b>
15.1	Basic Information . . . . .	123
15.2	Introduction . . . . .	123
15.3	Integrated Approaches for Modeling Transport and Emissions . . . . .	124
15.4	Emission Calculation . . . . .	125
15.5	Software Structure . . . . .	126
<b>16</b>	<b>Noise</b>	<b>129</b>
16.1	Basic Information . . . . .	129
<b>17</b>	<b>Accidents</b>	<b>131</b>
17.1	Basic Information . . . . .	131
<b>VI</b>	<b>Extending MATSim</b>	<b>133</b>
<b>18</b>	<b>Dynamic Transport Services</b>	<b>135</b>
18.1	Introduction . . . . .	135
18.2	DVRP Contribution . . . . .	136
18.3	DVRP Model . . . . .	137
18.3.1	Schedule . . . . .	137
18.3.2	Least-Cost Paths . . . . .	138
18.4	DynAgent . . . . .	138
18.4.1	Main Interfaces and Classes . . . . .	138
18.4.2	Configuring and Running a Dynamic Simulation . . . . .	139
18.4.3	RandomDynAgent Example . . . . .	139
18.5	Agents in DVRP . . . . .	140
18.5.1	Drivers . . . . .	140
18.5.2	Passengers . . . . .	140
18.6	Optimizer . . . . .	141
18.7	Configuring and Running a DVRP Simulation . . . . .	142
18.8	OneTaxi Example . . . . .	142
18.9	Research with DVRP . . . . .	143
<b>19</b>	<b>The “Minibus” Contribution</b>	<b>145</b>
19.1	Basic Information . . . . .	145
19.2	Paratransit . . . . .	145
19.3	Network Planning or Solving the Transit Network Design Problem with MATSim . . . . .	146
<b>20</b>	<b>Destination Innovation</b>	<b>149</b>
20.1	Basic Information . . . . .	149
20.2	Introduction . . . . .	149
20.3	Key Issues in Developing the Module . . . . .	150
20.3.1	Error Terms . . . . .	150
20.3.2	Quenched Randomness . . . . .	150
20.3.3	Search Space Construction and Evaluation . . . . .	152
20.3.4	Destination Choice Set Specification . . . . .	152
20.3.5	Facility Load . . . . .	154
20.4	Application of the Module . . . . .	155
20.5	The Module in the MATSim Context . . . . .	158
20.6	Lessons Learned . . . . .	158
20.7	Further Reading . . . . .	158

<b>21</b>	<b>How to Write Your Own Extensions and Possibly Contribute Them to MATSim</b>	<b>159</b>
21.1	Introduction . . . . .	159
21.2	Preparation: Scripts-in-Java . . . . .	160
21.3	MATSim Extension Points: General . . . . .	161
21.4	Extension Points Related to Scenario . . . . .	161
21.4.1	Customizable and ObjectAttributes . . . . .	161
21.4.2	Additional Scenario Elements . . . . .	162
21.5	Events . . . . .	162
21.6	Configuring the Controller (= inject syntax) . . . . .	163
21.6.1	Binding a Class (= an Implementation Type) . . . . .	163
21.6.2	Binding an Instance . . . . .	164
21.6.3	Binding a Provider (not important to get started) . . . . .	164
21.6.4	MATSim default bindings . . . . .	165
21.6.5	Advantages of the injection framework . . . . .	165
21.6.6	Pre-configured MATSim extension points . . . . .	165
21.6.7	ControllerListener: Handling Controller Events . . . . .	166
21.6.8	MobSim Listener . . . . .	167
21.6.9	TripRouter . . . . .	167
21.6.10	MobSim . . . . .	167
21.6.11	PlanStrategy . . . . .	168
21.6.12	Scoring . . . . .	169
21.7	Additional Config Groups . . . . .	169
21.8	MATSim extensions . . . . .	170
21.9	Conclusion . . . . .	170
<b>22</b>	<b>Organization: Development Process, Code Structure and Contributing to MATSim</b>	<b>171</b>
22.1	MATSim's Team, Core Developers Group, and Community . . . . .	171
22.2	Roles in the MATSim Community . . . . .	174
22.3	Code Base . . . . .	174
22.3.1	Main Distribution . . . . .	174
22.3.2	Core . . . . .	175
22.3.3	Contributions . . . . .	175
22.3.4	Playgrounds . . . . .	176
22.3.5	Contributions and Extensions . . . . .	177
22.3.6	Releases, Nightly Builds and Code HEAD . . . . .	177
22.4	Drivers, Organization and Tools of Development . . . . .	177
22.5	Documentation, Dissemination and Support . . . . .	178
22.6	Your Contribution to MATSim . . . . .	178
	<b>Acronyms</b>	<b>184</b>
	<b>Glossary</b>	<b>189</b>
	<b>Bibliography</b>	<b>193</b>

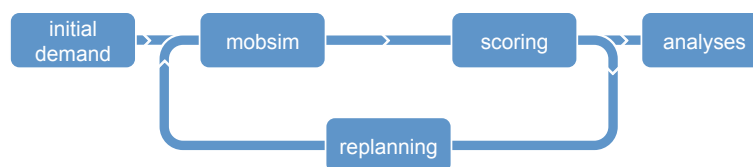


**Part I**  
**Using MATSim**



# Introducing MATSim

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



## 1.1 The Beginnings

The Multi-Agent Transport Simulation (MATSim) project (MATSim, 2016) started with Kai Nagel, then at ETH Zürich, and his interest in improving his work with the TRansportation ANalysis and SIMulation System (TRANSIMS) project (Smith et al., 1995; FHWA, 2013); he also wanted to make the resulting code open-source.<sup>1</sup> 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 Complex Adaptive Systems (CAS):

- **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".

<sup>1</sup>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' merger 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., 2012b) into the iterative loop.

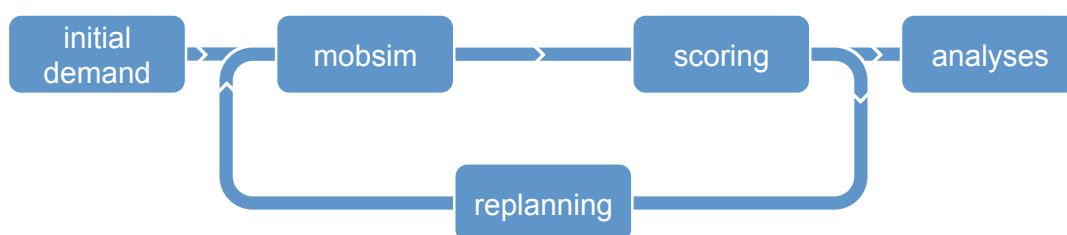


Figure 1.1: MATSim loop, sometimes called the MATSim cycle.

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,

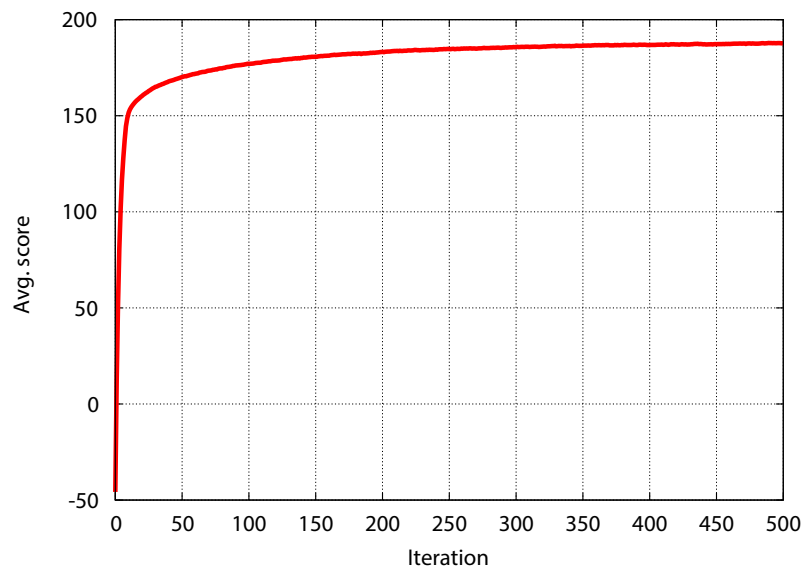


Figure 1.2: Typical score progress.

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 *mobility simulation* (*mobsim*) (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., 2011b, 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, 2012b). 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 10.

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

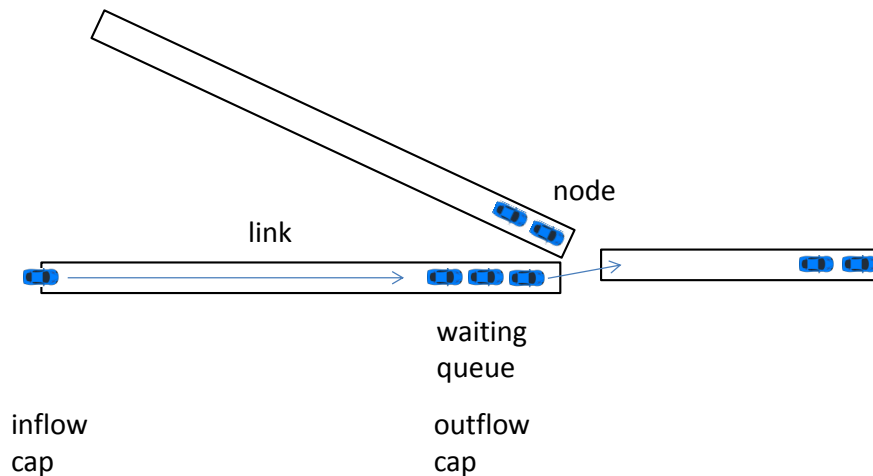


Figure 1.3: Traffic flow model.

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 21.5.

### 1.3 MATSim's Traffic Flow Model

MATSim provides two internal mobsims: QSim and Java Discrete Event Queue Simulation (JDEQSim); in addition, external mobility simulations can be plugged in. Some years ago, the Discrete Event Queue Simulation (DEQSim) 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

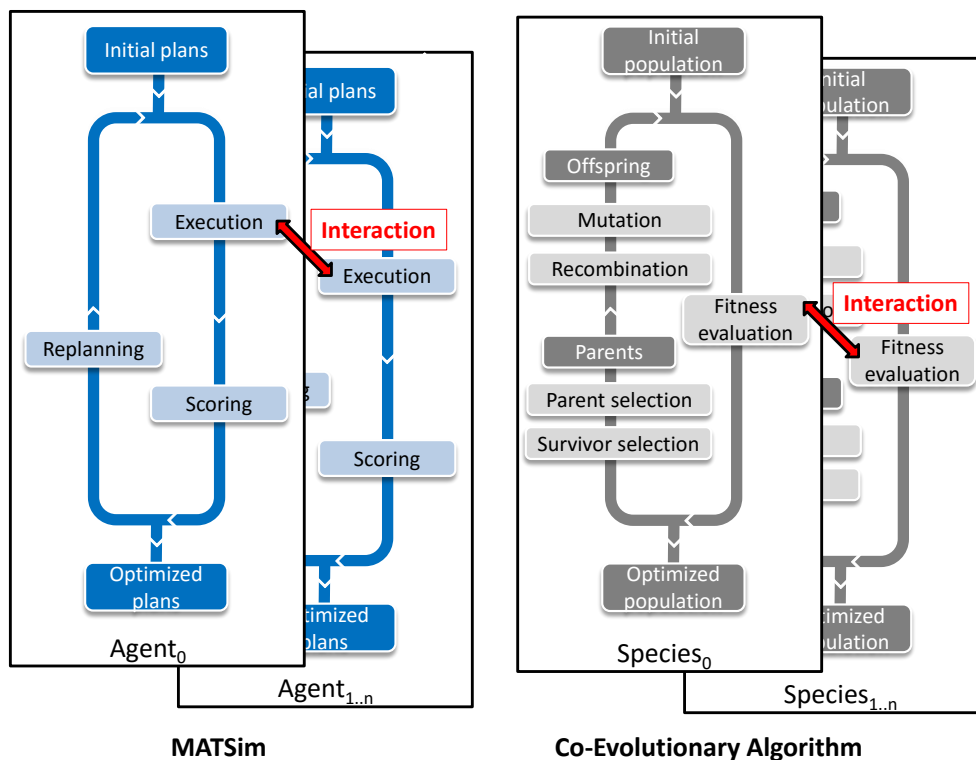


Figure 1.4: The co-evolutionary algorithm in MATSim.

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.

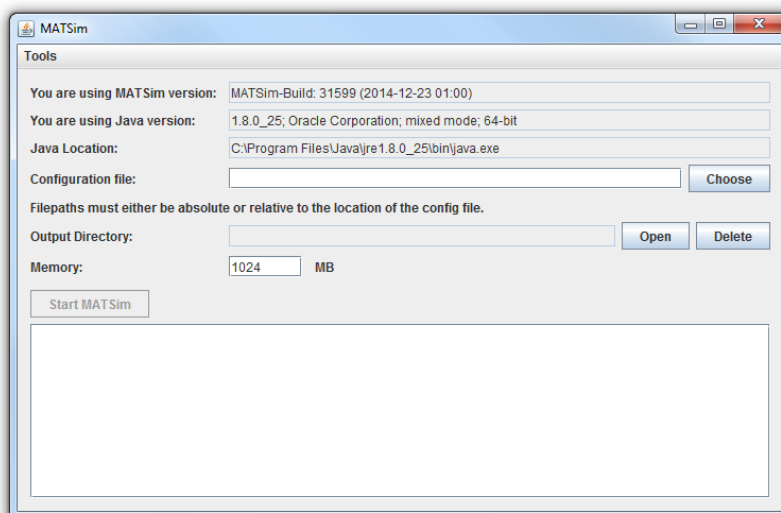
Note that there is a difference between the application of an evolutionary algorithm and a *co-evolutionary* algorithm. An evolutionary algorithm would lead to a system optimum, as optimization is applied with a global (or population) fitness function. Instead, the *co-evolutionary* algorithm leads to 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 Application Programming Interface (API) is described in Part II, Chapter 21.

## 2.1 Setting Up and Running MATSim

### 2.1.1 Setting Up MATSim

To run MATSim, you must install the Java Standard Edition (Java SE) that complies with the appropriate MATSim version. At this time (spring 2023), this is Java SE 17.

[MATSim example project on github](https://github.com/matsim-org/matsim-example-project) There is a so-called example project at <https://github.com/matsim-org/matsim-example-project>.

This version is targeted to programmers who are fluent with an Integrated Development Environment (IDE) (e.g. Eclipse, IntelliJ) and Java, and who want program against MATSim. The preferred workflow is:

1. Fork this project into your own workspace at GitHub.
2. Clone the forked project into your local IDE.

This will, for example, allow you to set up your own regression tests on GitHub.

The approach will automatically download MATSim (as a so-called Maven artifact), 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 code examples, along the lines discussed in Section 3.2.3, are provided at <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 Graphical User Interface (GUI) 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 into the **MATSim directory tree**.

**Maven** One can use MATSim as a 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** MATSim can be started by double-clicking the MATSim Java ARchive (JAR) file (since MATSim version 0.8). 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 a “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.<sup>1</sup>

<sup>1</sup>Via is commercial, see <https://www.simunto.com/via>. A free license is available for up to 500 MATSim agents. An alternative is On The Fly Visualizer (OTFVis) (Chapter 7), but it is not officially supported, and more difficult to use.

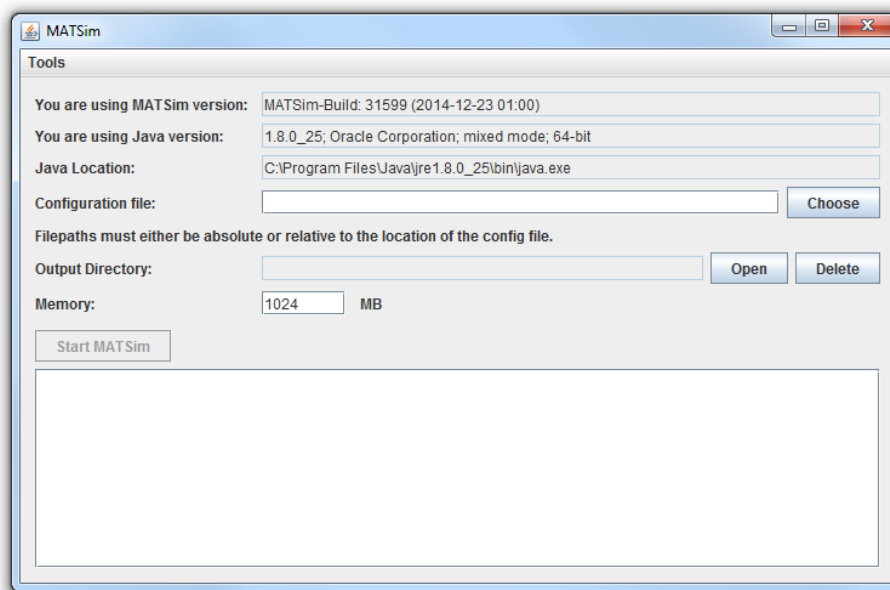


Figure 2.1: MATSim GUI.

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

**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 contains a list of settings that influence how the simulation behaves.

The structure of the config file is as follows:

- At the highest level, there are config groups. They are named “modules” in the config file:

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

or

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

- Inside each config group, there can be

- Key-value pairs, such as

```
<param name="firstIteration" value="0" />
```

- Parameter sets, such as

```
<parameterset type="activityParams">
  <param name="activityType" value="home"/>
  <param name="typicalDuration" value="12:00:00" />
  ...
</parameterset>
```

Parameter sets are typically used if there are more than one grouped sets of parameters of the same type. For example, there are several sets of activity parameters.

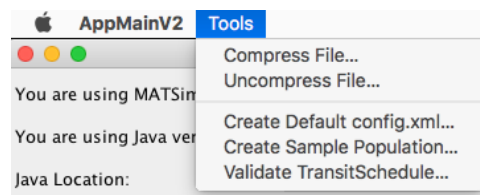


Figure 2.2: MATSim GUI tools.

The list of available parameters and valid parameter values keeps changing over time. 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.<sup>2</sup> 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 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 running only the initial (“zeroth”) iteration 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, see <https://github.com/matsim-org/matsim-example-project/tree/master/scenarios/equil/config.xml>.

There are two useful capabilities for file names:

- Most file names are relative to the directory of the config file.<sup>3</sup>
- Input file names can be given as URL.

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.

<sup>2</sup>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
```

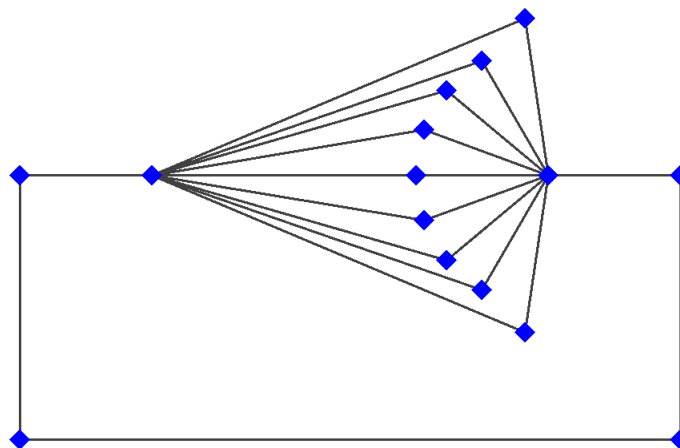
<sup>3</sup>Since MATSim version 0.9.x. The “most” refers to the fact that this needs to be manually implemented, and as a result this sometimes has not been done yet, in particular in contribs. If you encounter such a case, please request this feature via <https://matsim.org/faq>.

Further configuration possibilities are described in Chapter 4.

## 2.2 An Example Scenario

The MATSim example project comes with an example scenario named `equil` in the folder `examples/scenarios/equil`,<sup>4</sup> 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.



MATSim  
Multi-Agent Transport Simulation  
SENZOON

Figure 2.3: Equil scenario network.

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

**"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 (see Section 22.6).

```
<module name="strategy">
  <param name="maxAgentPlanMemorySize" value="5" />
  <!-- (0 means unlimited) -->
  <parameterset type="strategysettings" >
```

<sup>4</sup>See <https://github.com/matsim-org/matsim-example-project/tree/master/scenarios/equil>.

```

<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 10. 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>

```

"controle" 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="controle">
  <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 5) or OTFVis (Chapter 7).

## 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 Extensible Markup Language (XML) 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-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 2.3.4.3 and 8.1). 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 2.3.4).
- 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-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, `Controller` 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" />
    </plan>
  </person>
</population>

```



```
<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 Controller 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 controller 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.

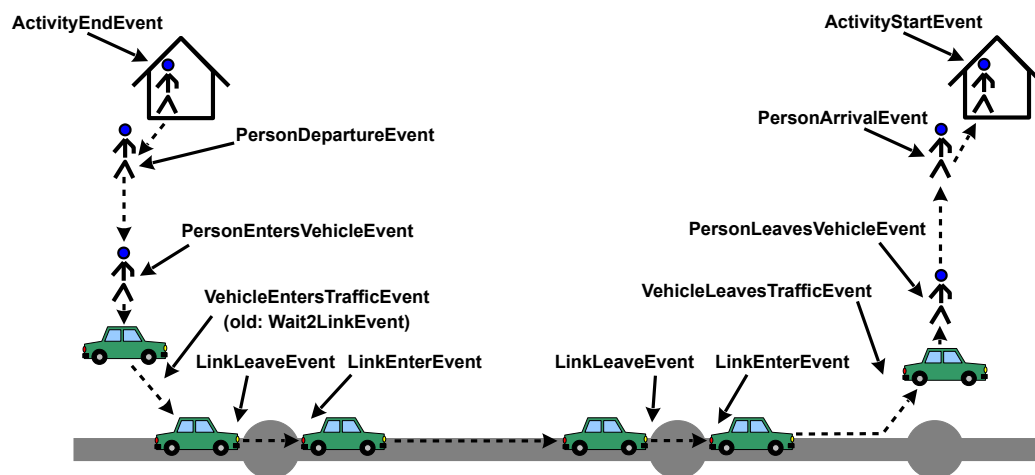


Figure 2.4: Mobsim events.

**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 21.5.

**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 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 10.4.

### 2.3.3 Data Requirements

### 2.3.3.1 Population and Activity Schedules

As discussed, MATSim needs a file with initial plans as input. All choice dimensions that will not be modified later through MATSim's day-to-day learning have to be exogenously estimated here. As of 2022, 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 often derived from surveys: for Switzerland, from the microcensus (Swiss Federal Statistical Office (BFS), 2006). Newer data sources, such as Global Positioning System (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.

### 2.3.3.2 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 OpenStreetMap (OSM), and commercial sources, including navigation network providers.

## 2.3.4 Units, Conventions, and Coordinate Systems

### 2.3.4.1 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 Universal Transverse Mercator (UTM) projected coordinate systems (see Section 2.3.4.3) 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

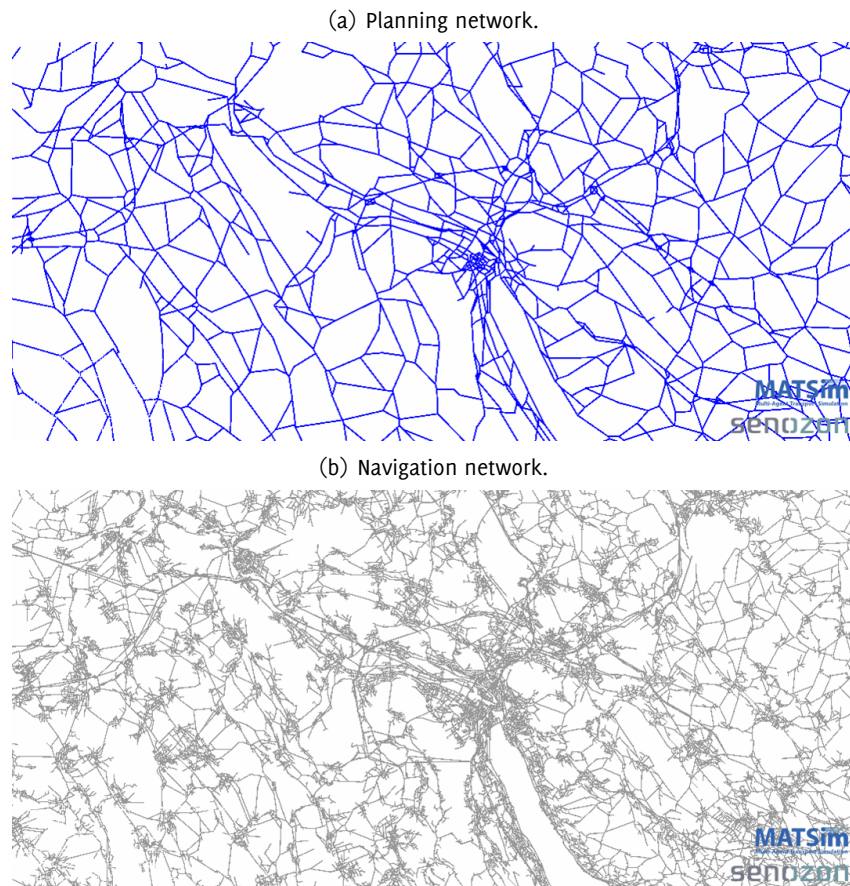


Figure 2.5: Zürich networks.

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 (10.5) below). Thus, when one moves from Germany to Switzerland, the parameter  $\beta_m$  must be changed from “utility per Euro” to “utility per Swiss Franc”.

#### 2.3.4.2 Conventions for Identifiers

MATSim uses IDs at many places. 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.

#### 2.3.4.3 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 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 locations of the code. Calculation of distances between spherical coordinates is 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 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 Geographic Information System (GIS) 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.

**Coordinate system for the MATSim run** 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 Quantum GIS (QGIS). 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 "European Petroleum Survey Group (EPSG) 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 Well-Known Text (WKT) format.

**Coordinate system for input files given in the input file** One can specify the coordinate system directly in the each input 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.

**Coordinate system for input files given in the config** 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.

### 2.3.5 Open Scenario Input Data

### 2.3.5.1 MATSim examples utils

A “trick” to get access to illustrative scenario data is something like the following:

```
URL context = org.matsim.examples.ExamplesUtils.getTestScenarioURL( "equil" );
URL url = IOUtils.extendUrl( context, "config.xml" );
Config config = ConfigUtils.loadConfig( url );
```

See `RunMatsimFromExamplesUtils.java` in `matsim-example-project`.

This works with the following entry in `pom.xml`:<sup>5</sup>

```
<dependency>
  <groupId>org.matsim</groupId>
  <artifactId>matsim-examples</artifactId>
  <version>${matsim.version}</version>
  <!-- <scope>test</scope>-->
</dependency>
```

Those lines are already included in `matsim-example-project`.

The scenarios can be found here: <https://github.com/matsim-org/matsim-libs/tree/master/examples/scenarios>.

### 2.3.5.2 Pre-packaged scenarios

Pre-packaged scenarios can be found under <https://matsim.org/open-scenario-data>.

That web page points to several places, one of them <https://github.com/matsim-scenarios>.

## 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 Uniform Resource Locators (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 10.4, 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`.

<sup>5</sup>This dependency should normally only be in the test scope, as indicated by the commented-out line. The reason is that otherwise it will be passed on to all inheriting projects. However, for a “leaf” project (on which no other projects depend), it is fine to have this in the main scope.

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 reduced config file.

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.*

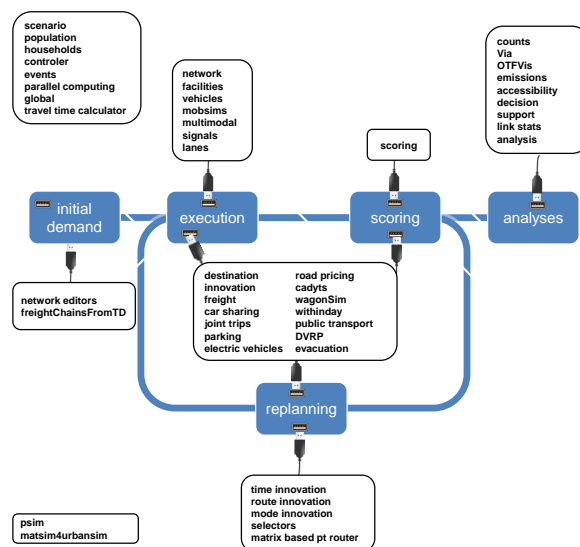
You should add them one by one and verify that they are doing what they should.





## 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 21 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;<sup>1</sup> 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,<sup>2</sup> are sometimes described as modules. It is important to understand the different levels of access stemming from the generally modular architecture.

<sup>1</sup> 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”.

<sup>2</sup>Standalone tools such as the MATSim network editor for Java Open Street Map Editor (JOSM), 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.
- Click on the MATSim jar file<sup>3</sup> 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 12).

### 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 controller object:  
    Controller controller = new Controller( scenario ) ;  
    controller.doZzz(...) ; // (**)  
    ...  
  
    // run the iterations:  
    controller.run() ;  
}
```

For a working example, see <https://github.com/matsim-org/matsim-example-project>; this will also contain a valid pom.xml, which pulls in all library dependencies including MATSim itself via Maven.

<sup>3</sup>This works since the 0.8.x release.

Extension points, especially at (\*) and (\*\*), are described in more detail in Chapter 21. 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.controller().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 ) ;
    Controller controller = new Controller( scenario ) ;
    controller.addOverridingModule( new OTFVisLiveModule() ) ;
    // *****
    controller.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 21, since this is the only way an extension can later become a contribution.

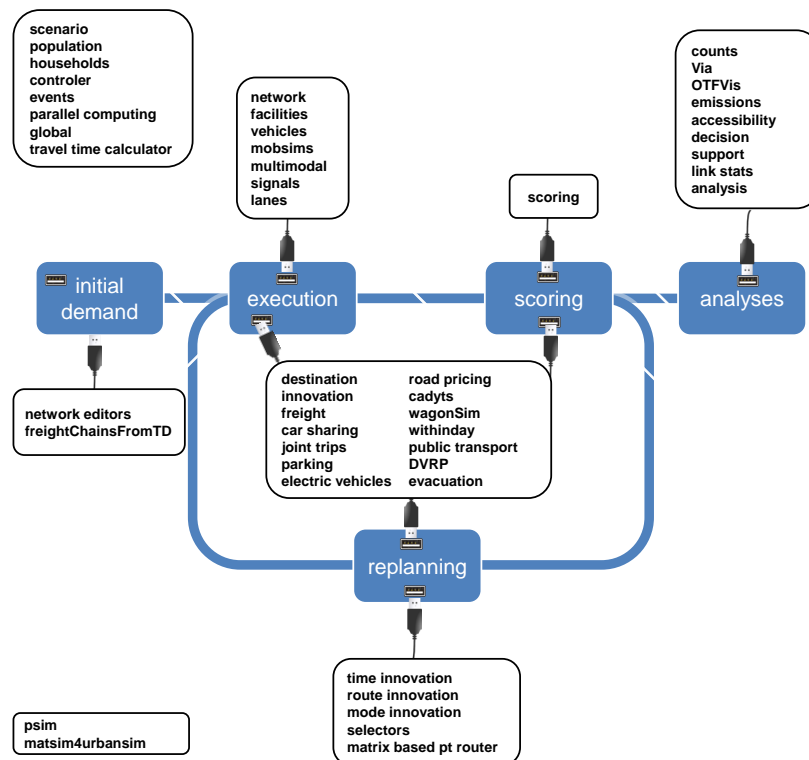


Figure 3.1: MATSim functionality.

### 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,<sup>4</sup> Verkehr In Städten - Umliegung (VISUM),<sup>5</sup> Equilibre Multimodal Multimodal Equilibrium (EMME), or Simulation of Urban Mobility (SUMO) (via the TraCI interface)<sup>6</sup> for plug-ins. SCALable LANGUAGE (Scala) 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 Technische Universität (TU) 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 22), 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.

Table 3.1: MATSim functionality overview.

Module	Described in	Comment
<b>Global Modules and Global Aspects</b>	Section 4.3	
Controler	Section 4.3.1	core
Events	Section 2.3.2 and 21.5	core
Parallel Computing	Section 4.3.2	core
Global	Section 4.3.3	core
<b>MATSim Data Containers</b>	Section 4.2 and Chapter 10	
Network	Section 4.2.1 and 10.2	core
Population	Section 4.2.2 and 10.3	core
Counts	Section 10.4	core
Facilities	Section 10.5	core
Households	Section 10.6	core
Vehicles	Section 10.7	core
<b>Network Editors</b>		
MATSim JOSM Network Editor	Chapter ??	rarely used?
Map-to-Map Matching Editors in Singapore	Chapter ??	rarely used?
The “Network Editor” Contribution	Chapter ??	rarely used?
<b>Observational Modules</b>	Section 4.7	core
Travel Time Calculator	Section 4.7.1	core
Link Stats	Section 4.7.2	core
<b>Scoring</b>	Section 4.5	core
<b>Basic Strategy Modules</b>	Section 4.6	
Time Innovation	Section 4.6.1.1	core
Route Innovation	Section 4.6.1.2	core
Mode Innovation	Section 4.6.1.3	core
Selectors	Section 4.6.2	core

*Continued on next page*

<sup>4</sup>[http://docs.qgis.org/testing/en/docs/pyqgis\\_developer\\_cookbook/](http://docs.qgis.org/testing/en/docs/pyqgis_developer_cookbook/)

<sup>5</sup>PTV (2011)

<sup>6</sup><http://sumo.dlr.de/wiki/TraCI>

Table 3.1 – Continued from previous page

<b>Mobsims</b>		
QSim	Section 4.4.1 and Chapter 11	core
JDEQSim	Section 4.4.2	rarely used?
Hermes	missing	maintained by SBB
<b>Individual Car Traffic</b>		
Signals and Lanes	Chapter ??	contrib; maintained by VSP; rarely used?
Parking	Chapter ??	
Electric Vehicles	Chapter ??	
Roadpricing	Chapter ??	contrib; maintained by VSP
<b>Other Modes Besides Individual Car</b>		
Public Transport	Chapter 12	core
The “Minibus” Contribution	Chapter 19	contrib; maintained by VSP
Semi-Automatic Tool for Bus Route Map Matching	Chapter ??	rarely used?
Events-Based Public Transport Router	Chapter ??	rarely used? should be integrated into core
Matrix-based pt router	Chapter ??	rarely used?
Multi-Modal Contribution	Chapter ??	functionality mostly integrated into core
Car Sharing	Chapter ??	contrib; maintained by M. Balac; superceded by newer package by same author?
Dynamic Transport Systems	Chapter 18	contrib; maintained by VSP
<b>Commercial Traffic</b>		
Freight Traffic	Chapter ??	contrib; maintained by VSP
<b>Additional Choice Dimensions</b>		
Destination Innovation	Chapter 20	contrib; the “frozen epsilons” material is maintained by VSP
Joint Trips and Social Networks	Chapter ??	
<b>Within-Day Replanning</b>		
Within-day Replanning	Chapter ??	mostly integrated into core
Belief Desire Intention (BDI) Framework	Chapter ??	maintained by RMIT
<b>Automatic Calibration</b>		
Cadyts integration	Chapter ??	contrib; maintained by VSP
<b>Visualizers</b>		
Via Visualizer	Chapter 5	commercial; maintained by simunto
OTFVis Visualizer	Chapter 7	contrib; maintained by VSP
<b>Analysis</b>		contrib; material is not well organized
Accessibility	Chapter ??	contrib; somewhat maintained by VSP
Emissions	Chapter 15	contrib; maintained by VSP
Interactive Analysis and Decision Support	Chapter ??	rarely used?
The “analysis” contrib	Chapter 14	

Continued on next page

Table 3.1 – Continued from previous page

<b>Computational Performance Improvements</b>		
PSim	Chapter ??	rarely used?





# 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.

The config can evidently be modified by modifying the config file. It can also be modified in code. The basic syntax is given in Sec. 3.2.3. If you are planning to use MATSim through an IDE, then this is often the better option. In the following, that syntax is not separately explained, but in many cases it should be straightforward for figure it out from the XML syntax together with tab completion in the IDE.

### 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 10.2.

#### 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 10.3.

### 4.2.3 Further MATSim containers

Further MATSim containers are described in Chapter 10.

## 4.3 Global Modules and Global Aspects

### 4.3.1 controller config file section

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

```
<module name="controller" >
  ...
</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. If total RAM memory is not an issue, then 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 2.3.4) 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

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.

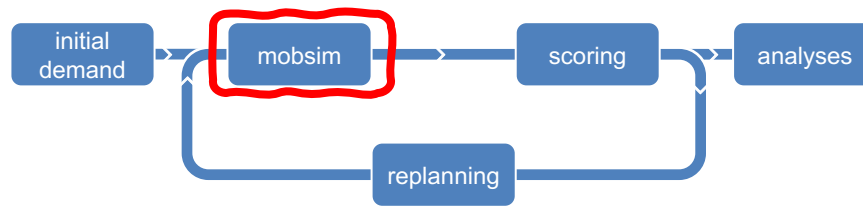


Figure 4.1: Mobsim within MATSim cycle.

**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 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 `controller` config file section must be set to JDEQSim, and a config file section

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

must be provided.

#### 4.4.3 Hermes

Hermes was commissioned by SBB (Swiss Federal Railways) and developed by ETH Zurich. Its purpose is to concentrate on the most important 80% of the functionality, and provide a faster implementation for that. It is maintained and used a lot by SBB. It is less appropriate for research, since it rarely contains the newest features with respect to a certain problem.

## 4.5 Scoring

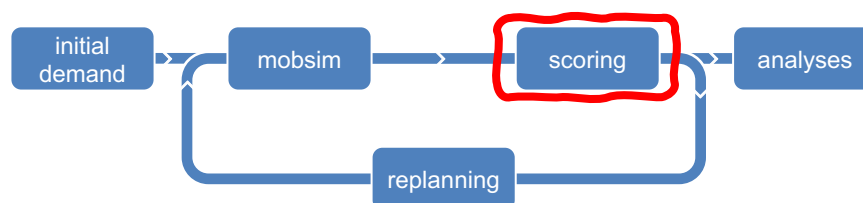


Figure 4.2: Score within MATSim cycle.

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 10.

## 4.6 Replanning Strategies

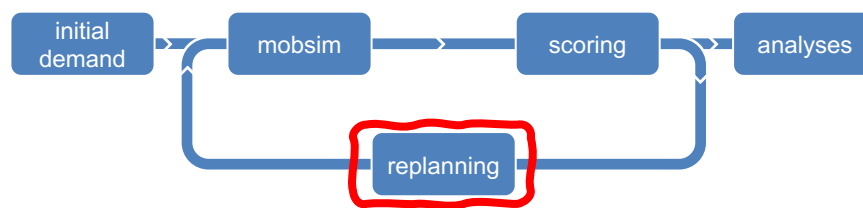


Figure 4.3: Replanning within MATSim cycle.

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="ChangeExpBeta" />
    <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.

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" />
  
```

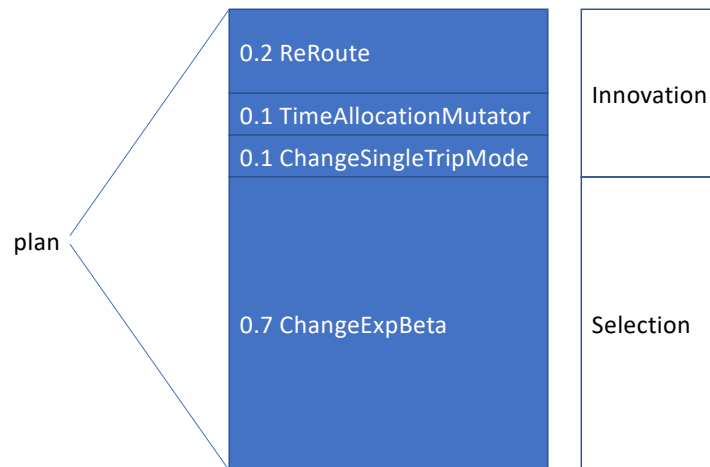


Figure 4.4: Replanning according to the above XML-specification.

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 controller 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.<sup>1</sup>

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.

<sup>1</sup>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.

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.

## 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 Multinomial Logit Model (MNL) selection between plans. It can be configured by the BrainExpBeta parameter from the scoring config group<sup>2</sup> 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  $\Delta_{score}$  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.<sup>3</sup> This selector has never been investigated systematically.

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.

<sup>2</sup>This is in the scoring config group for historical reasons.

<sup>3</sup>Also in the scoring config group for historical reasons.



### 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 Method of Successive Averages (MSA) averaging of the scores (Section 10.3.3).

## 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. As one of their many functions, link stats are used for comparison with count values, as introduced in Section 10.4.

## 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 under <https://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.<sup>4</sup>

---

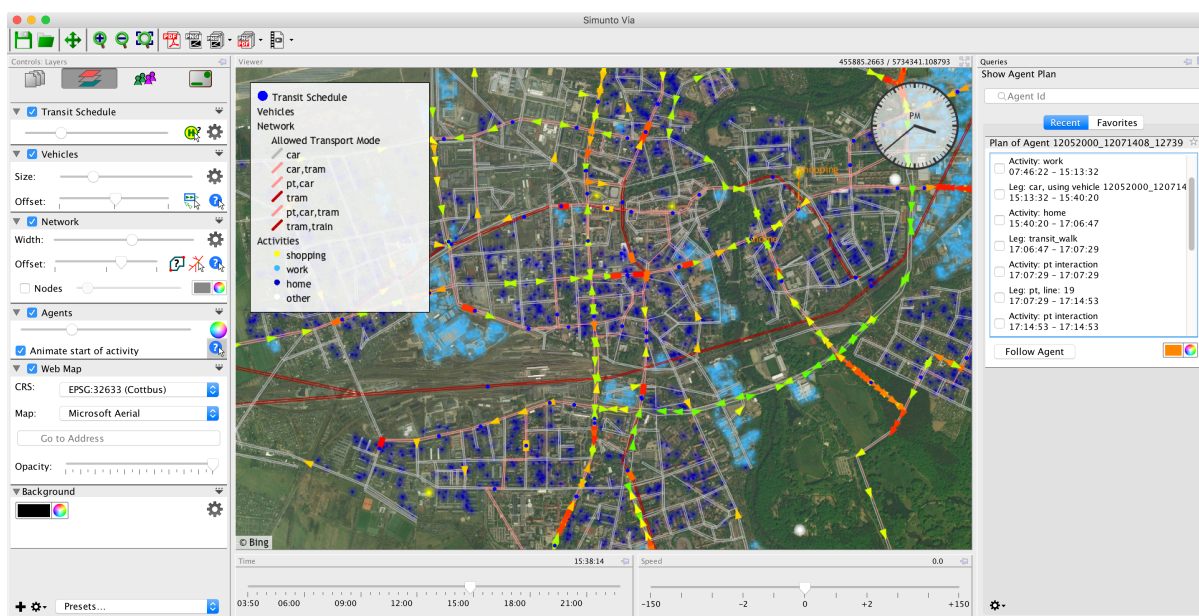
<sup>4</sup>Some additional material from that file still has to be removed manually – we are not there yet in fully cleaning this up.



**Part II**  
**Visualizers**

# Simunto Via

Author: Marcel Rieser



## 5.1 Basic Information

Entry point to documentation:

<https://simunto.com/via>

Invoking the module:

Standalone GUI, double-clickable jar file

Selected publications:

<https://www.simunto.com/via> → Download → manual

## 5.2 Introduction

*Via* is an application to visualize and analyze MATSim simulation results. Unlike MATSim, *Via* is not open source; it is developed as a proprietary commercial software by Simunto GmbH, a company founded

by a former PhD student involved in MATSim development. The development was originally started by Senozon AG, an Eidgenössische Technische Hochschule (ETH) Spin-off company working with MATSim. Shortly after Senozon AG was founded, first (potential) client presentations began; the lack of visual material was an obvious handicap. Explaining to customers that all answers to their questions were contained in a huge events file was not satisfactory; pictures or even animations made it much easier for them to understand. Thus, work on a visualization tool started as soon as the company was set up. Initially planned as a purely internal tool, it quickly became clear that a graphical visualization and analysis tool would also benefit other users of MATSim. After a beta test phase with selected MATSim users in Spring 2011, the first version of *Via* was released in July 2011. Since then, the list of features provided by the application has grown continuously. In early 2018, ownership of *Via* was transferred to Simunto GmbH which continues to enhance and support the application.

*Via* is written in Java and thus works on any platform able to run MATSim. For easier deployment, the application comes as double-clickable, native executable on Windows and Mac OS X, partially hiding its Java nature. A limited version is available for free and can be downloaded from the product website (Simunto GmbH, 2018). Different licenses are available for commercial usage or for research or educational purposes to serve different user group needs.

*Via* includes some general functionality that most people will use in the core application, like visualizing networks, facilities, vehicles and activities. Optionally available plugins provide additional features often relevant only to specialized user groups. This includes functionality related to public transport, comparison with car counts, using web maps like Google Maps or OSM as background, aggregation analyses, or movie recording.

*Via* allows customization of its window. The following descriptions refer to elements as they are placed in the default layout. The default configuration can be re-created by choosing *Reset Window State* from the *Window* menu in *Via*.

## 5.3 Simple Usage

*Via* differentiates between data sets, and how the data is visualized. It does so by managing data sources (typically MATSim files like `network.xml` or `events.xml`), and layers (e.g., displaying the network, vehicles, activity locations). A layer can use more than one data source for its visualization purposes (e.g., a network and some data from the events), and a data source can be used by multiple layers (e.g., events can be used by many different layers to visualize different things like vehicles, activities, link volumes, etc).

By default, *Via*'s window looks similar to the one shown in Figure 5.1. To add a file as a data source, the file can either be drag-and-dropped onto the layers list left of the black visualization area, or by choosing *Add Data...* from the *File* menu. To add a layer, the little plus icon in the lower left of the window can be pressed, or by choosing *Add Layer...* from the *File* menu. To get started, it's usually best to add a network and (small) events file from MATSim to *Via*, and create a *Network* layer and a *Vehicles* layer.

Elements shown in the visualization area like the network or vehicles can be queried. Queries are usually provided by layers, made available with buttons with question-mark icons. Clicking such an icon activates the corresponding query mode, and any subsequent click on the visualization area will run the query. Query results are shown on the right side of the visualization area. Figure 5.1 shows a network query for links. One query is special, globally available, and not linked to a layer: querying an agent plan. This query is available from the toolbar, next to the icon, to shift the visualization view around.

Once a query has been made, *Via* often allows another query based on the current query results. By right-clicking in the visualization area, a pop up menu appears with more options regarding the last query, as well as additional possible queries. Examples are: *Select Link Analysis*

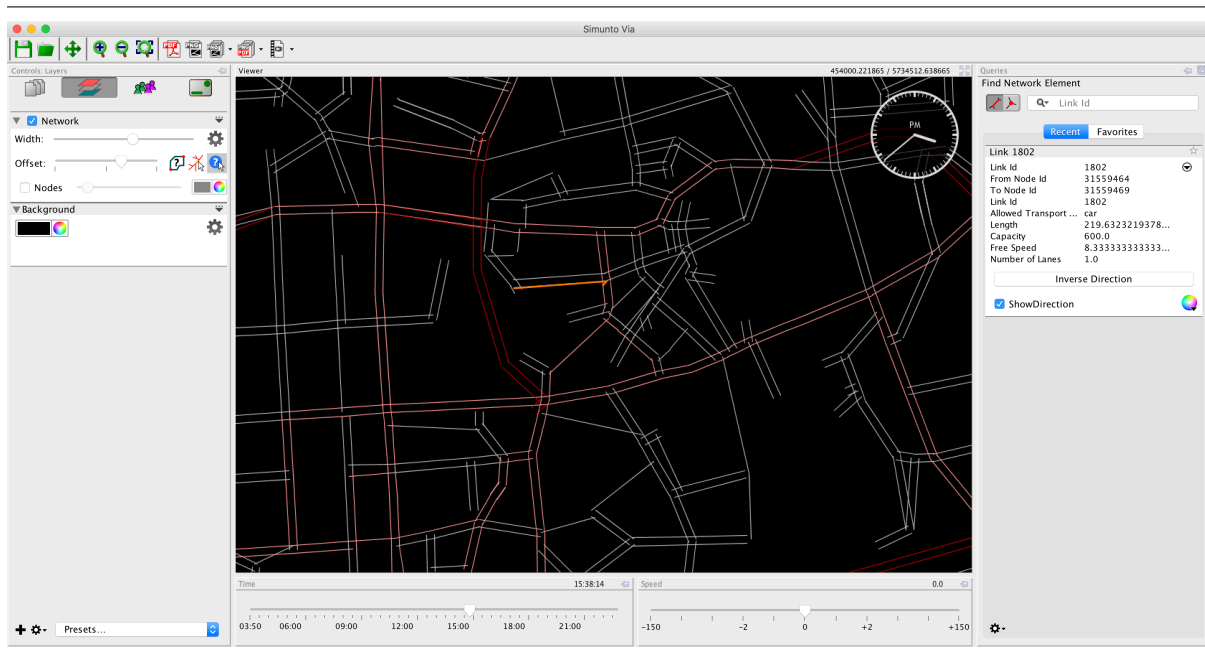


Figure 5.1: Via's window with default layout and a network query being shown.

given a link, Select Facility Analysis given a facility, List Transit Lines that use a given link, or List Passengers if a transit vehicles was queried in the first place.

## 5.4 Use Cases and Examples

### 5.4.1 Agent Visualization

The animated visualization of agents moving around in the modeled area was one of the main features in *Via*'s original development. To do this, *Via* needs only the `network.xml` and `events.xml` files from a MATSim run as data sources. For the visualization, a Network layer, Vehicles layer and activities layer must be created. With this setup, vehicles will move around in the visualization area as time progresses, and agents performing activities will be represented as colored dots.

The visualization can be further customized; with the addition of a `population.xml` file, more detailed activity coordinates can be loaded to obtain a better distribution of activity locations (MATSim's events file does not contain coordinates for activities, only the assigned link ID. So by default, all activities taking place on a link are first shown at the location of the link's to-node). Vehicles and groups of vehicles can also be styled differently; it is possible to visualize transit vehicles with a square shape with colors representing the occupancy of the vehicles, pedestrians or cyclists in a multi-modal simulation can be shown as circles and private cars can be displayed with a triangular shape with colors representing their absolute speed or their speed relative to the allowed maximum speed on their current link (see Figure 5.2). As mentioned above, arbitrary groups of vehicles can be styled differently, which is useful to highlight special agents, e.g., when simulating a fleet of electric vehicles, a car sharing fleet, or agents simulated with special routing guidance.

It is also possible to load arbitrary attributes for agents and then use those attributes for visualization purposes, e.g., having different colors for vehicles driven by agents who are employees, have a high income or are within a certain age range.

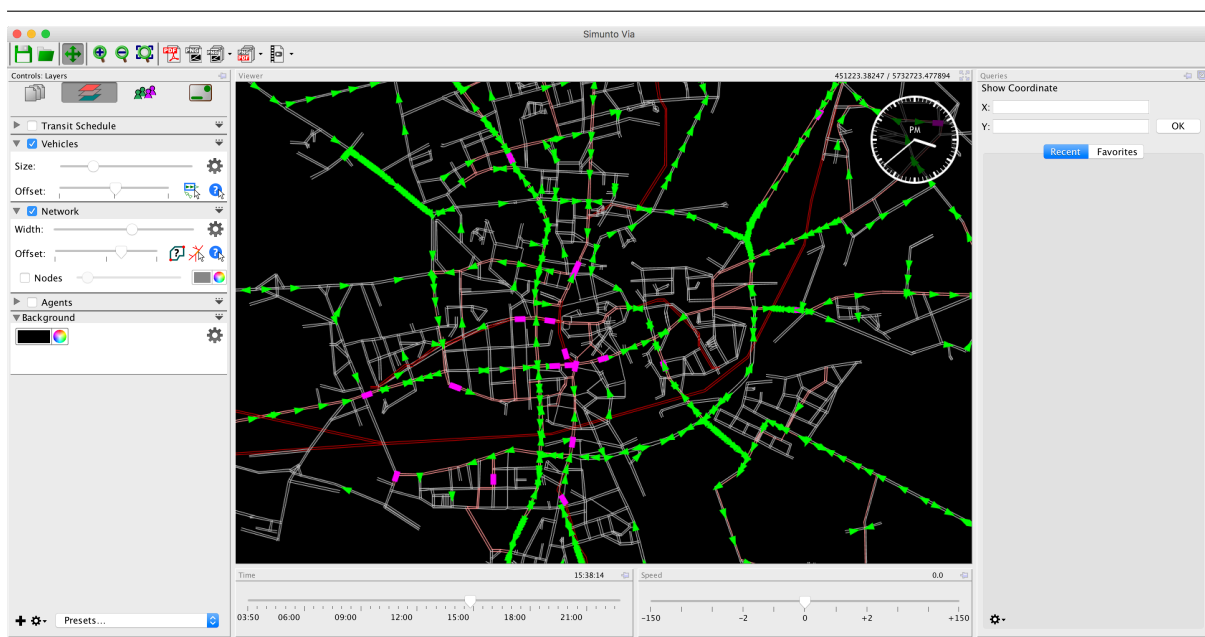


Figure 5.2: Vehicles in Via: Green triangular symbols represent private cars, pink rectangular symbols public transport vehicles.

### 5.4.2 Facility Analysis

Activity facilities allow for very detailed modeling in MATSim, especially considering the functionality provided by the destination innovation module (Chapter 20). *Via* provides several unique ways to analyze the mobility effects to and from facilities.

For each facility, a detailed analysis can be performed showing the number of agents arriving at, departing from, or staying at a facility over the simulated time. The numbers can be differentiated by the type of activity the agents perform at the facility, by the transport mode they arrive or depart with, or by other arbitrary agent attributes loaded by users.

An alternative analysis is similar to the – for transport planners – well known Select Link Analysis, but designed for facilities: the Select Facility Analysis. This analysis shows the combined link loads produced by agents arriving or departing at a facility, showing the starting location for agents visiting a specific facility and what routes they use.

### 5.4.3 Public Transport Analysis

The public transport plugin provides many different functions for analyzing public transport simulations. It starts with providing the specified vehicle types as agent attributes, so the vehicles can be differently visualized, based on the vehicle type they represent. Also, the absolute or relative occupancy of a transit vehicle is provided as attribute, allowing transit vehicles to be visualized accordingly. For stop locations, the number of passengers waiting for a bus or train can be plotted over the time of day, and the occupancy along a bus or train route can be visualized.

A special, but very useful visualization is the Route Flow analysis. This shows, in a visually appealing way, the number of passengers traveling between two stops along a route—for all possible stop combinations. Figure 5.3 shows an example of such a route flow with the route of the transit line shown in the background. It is clear that the demand on the bus route is more or less split in two; a first travel demand up to about the first third of the route, and then it again collects passengers all wishing to go to one of the last stops along the route. This could indicate that it might make sense to split the line in two.

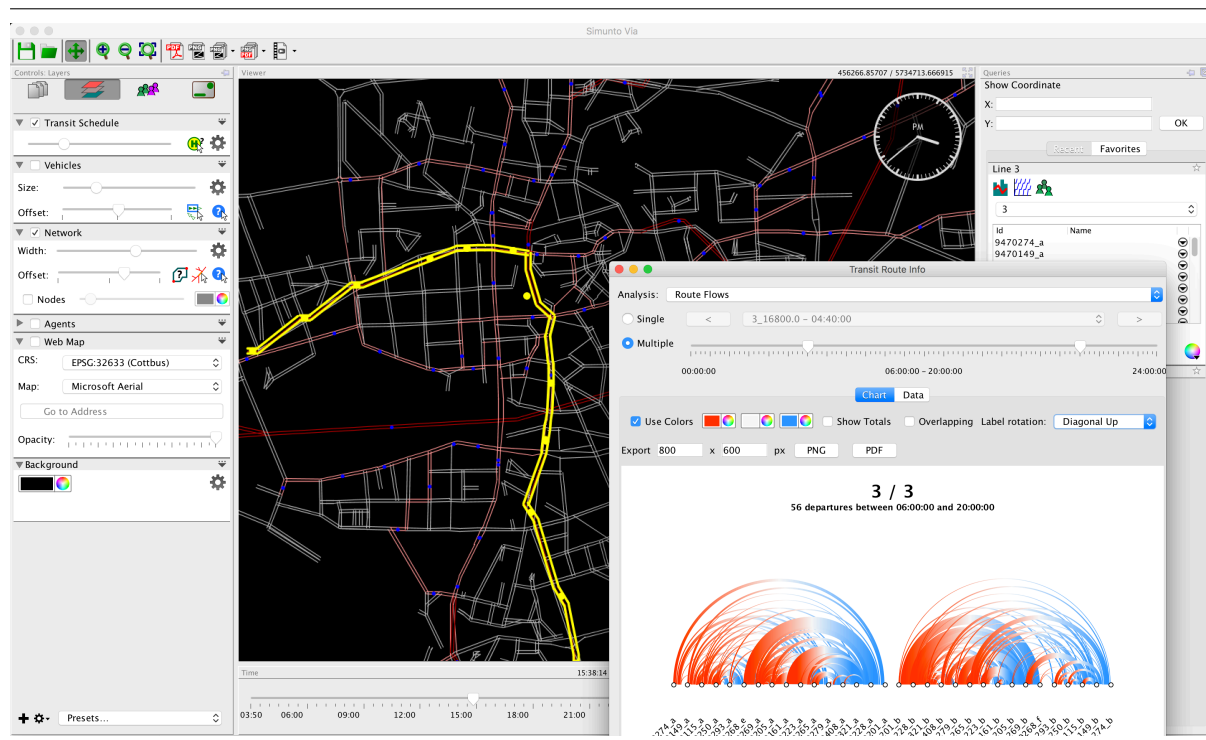


Figure 5.3: Passenger flows on a transit line.



#### 5.4.4 Scenario Comparisons

A typical use of MATSim is simulating a base case and then one or more case studies. Comparing scenarios then becomes an important step in the analysis of the different case studies. *Via* allows comparison of the link volumes of two scenarios visually by coloring the network with the absolute or relative difference of the link volumes between two models. In the future, other differences like average speeds will be supported too. The differences are time-dependent, aggregated over time intervals as small as 15 minutes.

#### 5.4.5 Aggregating Data

While MATSim requires and produces a lot of disaggregated data, it is still often necessary to aggregate data to make statements or predictions about a simulated scenario. *Via* provides a powerful mechanism to easily build arbitrary aggregations of available data. Such data can be either point data (like activity locations, trip start locations, GPS points or any other spatial point data) or origin-destination data (like trips with a start and end location, or the relation of an activity location to the home location of the agent performing the activity). While *Via* provides: activity locations, trip start and trip end locations, facility locations (automatically) as point data sources for aggregation, and the trips performed by agents as Origin-Destination (O-D) data sources, any tabular custom data with coordinate attributes can also be used for this.

Data can be aggregated into a rectangular or hexagonal grid, where the cell-size can be specified by the user, or into arbitrary zones provided as Environmental Systems Research Institute (ESRI) shape file by the user. The data points can be filtered by any of the available attributes, and the aggregation can either just count the data points in each region, or build the sum, the minimum or maximum or average of a data points attribute.

With the activity locations provided by an `Activities Layer`, the following (and more) aggregations are possible:

- show number of performed activities per region,
- show number of performed work activities per region,
- show number of work activities starting after 10 am per region, and
- show average duration of work activities starting after 10 am per region.

Similarly, with trip data provided by a `Vehicles` layer, the following exemplary aggregations are possible:

- show number of trip starts per region,
- show number of trip starts with mode “car” per region,
- show percentage share of trips starting with mode “car” in a region, compared to all trips starting in that region, and
- show average duration of trips starting with mode “pt” in a region after 11 am.

By using custom data tables, e.g., containing more information about trips, i.e., the ‘from and to’ activity types they connect, the number of line switches if it is a public transport trip (this requires the aggregation of MATSim’s legs to trips for analysis purposes), many more complex analyses are just a few clicks away in *Via*, like showing the average duration of car-trips starting between 6 am and 8 am, going from “home” to “work”.

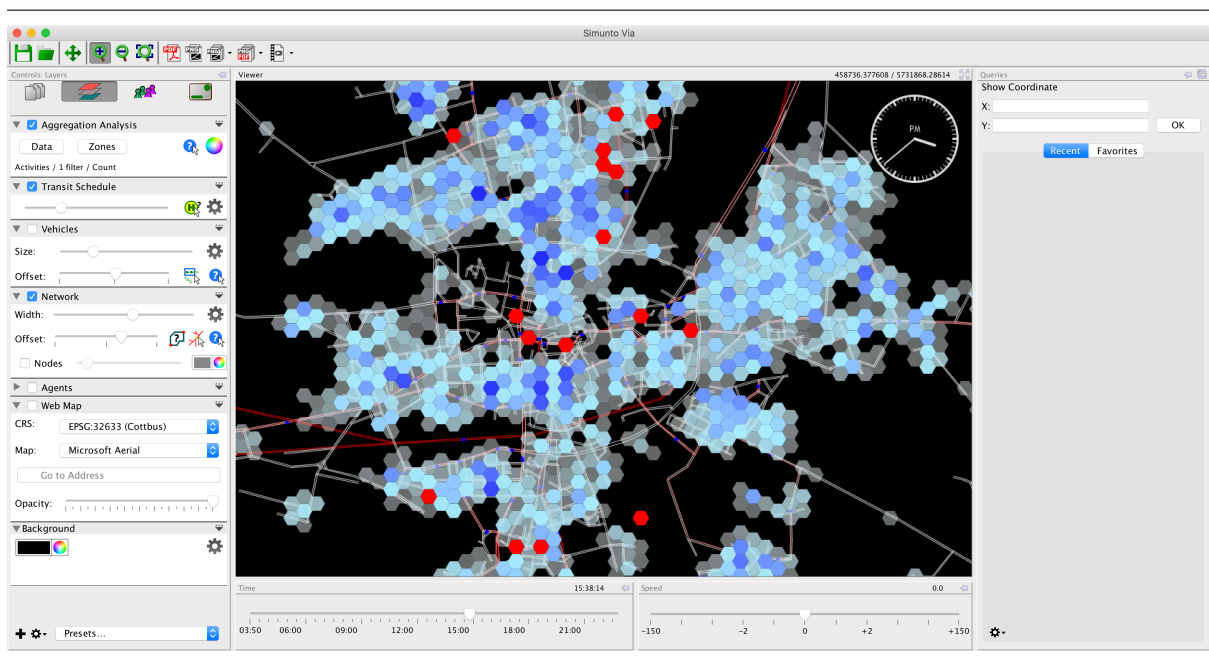
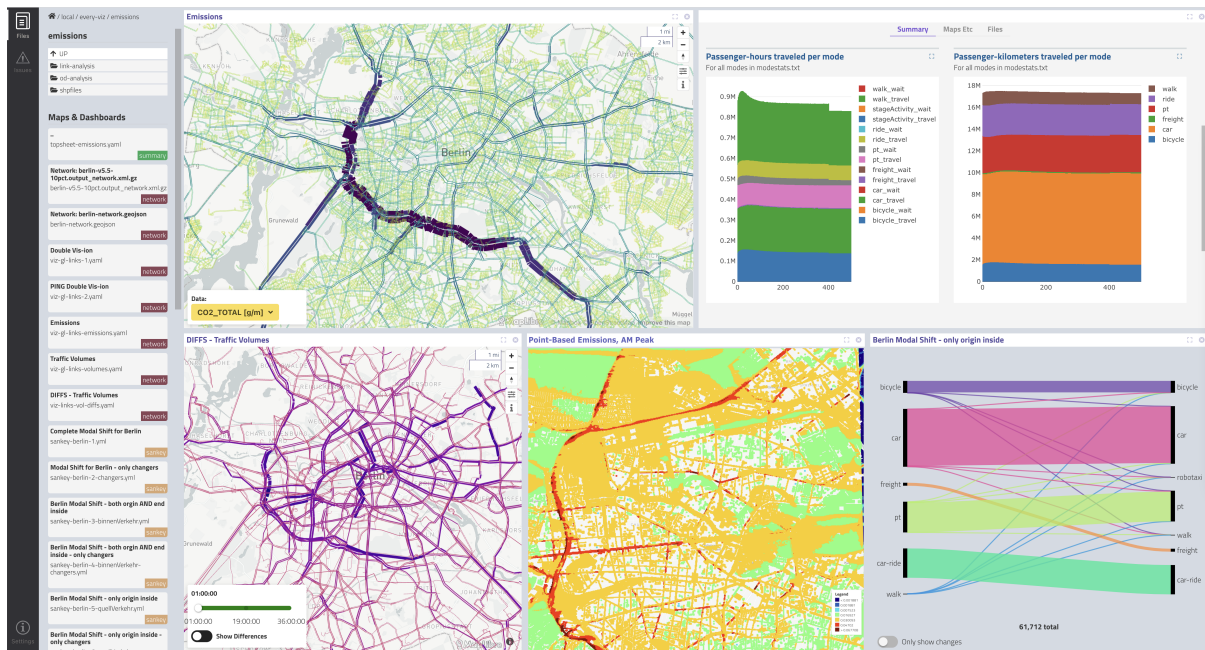


Figure 5.4: Aggregation analysis: Number of performed activities during the whole day.

# 6

## SimWrapper

Author: Billy Charlton



### 6.1 Basic Information

Entry point to documentation:

Documentation website: <https://simwrapper.github.io/docs/intro>

Invoking the module:

SimWrapper application available at <https://vsp.berlin/simwrapper>

Selected publications:

Charlton and Sana (2022)

## 6.2 Introduction

This chapter describes SimWrapper, an open-source web-based data visualization platform which can produce many different types of data visualizations based on MATSim outputs. These visualizations include roadway network volume plots, transit network summaries, and aggregated views of origin/destination patterns, and many others. The various types of visualizations can also be combined into pages known as dashboards, for interactive exploration.

SimWrapper, in a nutshell:

- is a website that runs client-side javascript in the form of a “single page application”, a common approach in current web development that is compatible with all modern web browsers;
- allows the user to navigate through their local filesystem or shared network storage of model runs to view results that are saved in a specific folder, rather than a database-centric approach. This matches the design of MATSim and other simulation models which produce collections of output files by default;
- provides a collection of data visualization archetypes that are each appropriate for displaying a certain type of data, for example various statistical chart types (bars, lines, area, pie), geographic data viewers supporting road and transit network link data, area aggregation (“choropleth” and “spider”) maps, XY coordinate plots, and many more;
- supports network-based file storage for public- and/or group-accessible shared data files (model runs), but has no other back-end server requirements and can run completely locally if no network file storage is available or needed;
- can combine all of these disparate components into cohesive dashboards that the user can lay out in a flexible manner, using small declarative configuration files. These configurations can be applied across multiple projects or simulation runs;

SimWrapper is evolving rapidly, and this book cannot keep up with the latest features in perpetuity. Users interested in using SimWrapper for MATSim analysis tasks are directed to the main SimWrapper documentation website at [simwrapper.github.io](https://simwrapper.github.io). The main website includes the latest guides for getting started with the tool, as well as detailed reference details for the many visualization types available in the tool.

## 6.3 Starting SimWrapper

Browse to <https://vsp.berlin/simwrapper> to access the VSP instance of the SimWrapper website.

If your simulation results are on your local machine, you can immediately access the files via Google Chrome and Microsoft Edge by selecting “Add local folder” from the main SimWrapper homepage and authorizing access to a local folder.

*Not using Chrome?* If you are not using Chrome or Edge browsers, you can still access local folders by running the small helper Python script available at <https://pypi.org/project/simwrapper/> which runs a private local webserver on your machine. To avoid this complication, we recommend using Google Chrome.

If your simulation results are on network file storage, work with your network administrator to provide HTTP access to the files. This is explained on the SimWrapper website at <https://simwrapper.github.io/docs/file-management>.

To work with files on SSH-accessible computing machines including the TU Math Cluster, you must install SSHFS for your operating system and mount the cluster machine files to a local folder. Then proceed as above.

## 6.4 Using SimWrapper to view MATSim results

The high-level workflow is as follows: after running a simulation, some outputs such as MATSim trips, networks, and events can be directly viewed in SimWrapper, while other outputs require some post-processing to produce summary datasets in Comma-Separated Values (CSV) comma-separated value format.

Users are not expected to know or use JavaScript, Python or any other notebook programming language; SimWrapper is essentially just a website like any other. It reads the users files directly and builds visualizations according to the available files and configuration in the model output folders.

### 6.4.1 SimWrapper views based on regular MATSim output

The following outputs are viewable in SimWrapper without any post-processing or additional effort, simply select the output file to open it in an interactive viewer:

- **Network** - The MATSim output network can be loaded, and then datasets can be "joined" by link IDs to produce useful visualizations
- **Transit** - Transit output networks can be viewed and various routes explored interactively. If there is passenger ridership information embedded in the file it can also be viewed.
- **Output Trips** - The standard output\_trips trip origin/destination file can be viewed as origins and destinations aggregated to various sizes.
- **Carriers** - Freight and other carrier outputs including views of shipments, routes, and depots are visualized
- **Events** - Vehicles on links can be visualized across time of day using the event viewer. Note that web browsers have fundamental data size and memory constraints, more so than other desktop software, thus some event files may be too large for your browser to ingest.

### 6.4.2 SimWrapper dashboard generated by the SimWrapper contrib

An easy way to generate a SimWrapper dashboard is to add

```
controller.addOverridingModule( new SimWrapperModule() );
```

into Controller as shown in Section 3.2.4. This will prepare the output directory (as defined in the config) so that opening the directory in SimWrapper (as described in Section 6.3) will bring up a meaningful SimWrapper dashboard.

### 6.4.3 Self-configured SimWrapper dashboards

Beyond these views that are present "out of the box," SimWrapper can also be configured to display many different types of data:

- **Area maps** - Data that is aggregated into districts, zones, etc, can be viewed in area maps, with extensive symbology support
- **XY/Time plots** - Any tabular data that has a coordinate and a time point for each row can be explored by time of day. This is useful for point-based emission and noise data, for example
- **Statistical charts** - A large number of charts and plots such as area, bar, scatterplot, pie, and more are all available through integration with well-known Plotly and Vega-Lite charting tools.

- **Summary tables** - A data summarization engine is embedded in SimWrapper that allows you to produce top-level summary statistics from data tables and format them in an output panel.

For this second set of visualizations, some work is required by the user: regular MATSim outputs need to be post-processed to produce tabular data that the user wishes to view. This is beyond the scope of this chapter; often, the Java code for running the simulation can produce these outputs directly, and if not then Python or R scripts could be written to do the necessary processing. The scripts depend heavily on what you wish to view. Refer to chapter V for an introduction to these tools.

## 6.5 Building interactive dashboards with SimWrapper

One of the key features of SimWrapper is its ability to lay out multiple visualization panels in a single cohesive page, known as a dashboard. All of the visualization types described above can be laid out as panels in a SimWrapper dashboard.

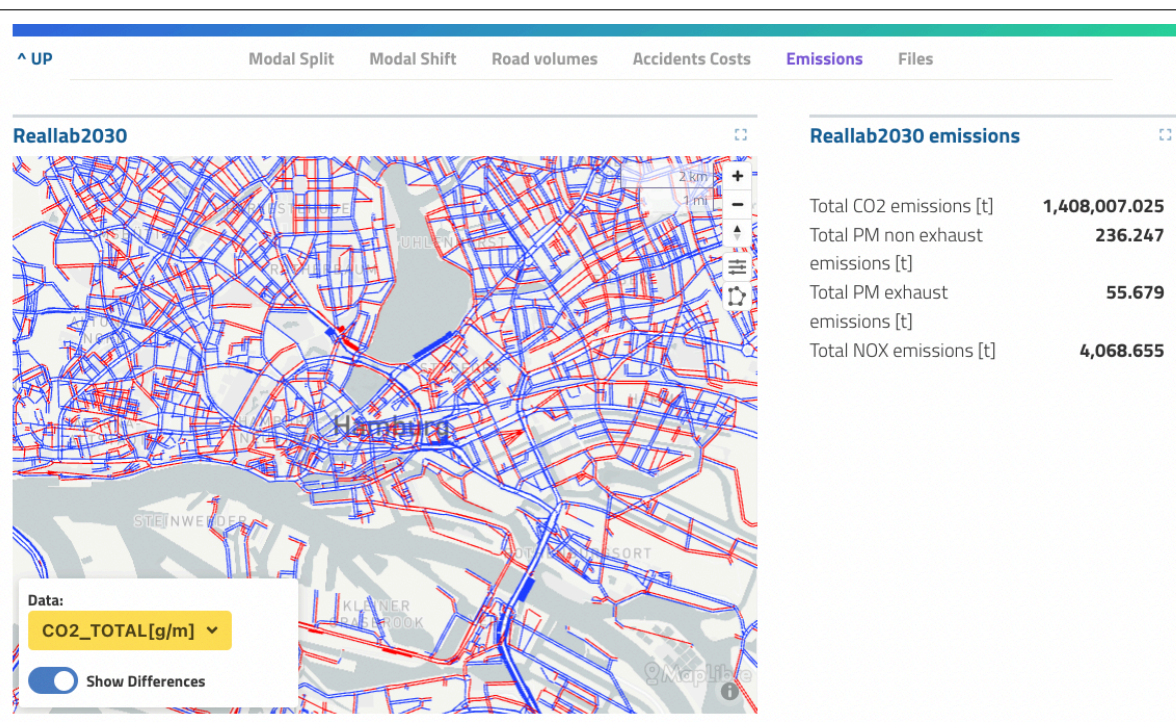


Figure 6.1: Example SimWrapper dashboard.

A dashboard is defined using a Yet Another Markup Language (YAML)-formatted configuration file, which describes the layout and content of the dashboard and its constituent panels. Along with the input data, the user provides configuration details for each dashboard which define the layout and any additional parameters. Typical configuration details are the names and locations of input files, color and width symbology specifications, and so on. These parameters can be set on a project level or for individual runs, and are stored as specially-formatted text files using the common “YAML” text markup format. The website reads these files and generates the dashboards. Dashboards can be organized as separate pages, be full-screen, or use a special side-by-side mode for comparison tasks.

## 6.6 To learn more

SimWrapper is evolving rapidly, and this book cannot keep up with the latest features in perpetuity. Users interested in using SimWrapper for MATSim analysis tasks are directed to the main SimWrapper documentation website at [simwrapper.github.io](https://simwrapper.github.io).

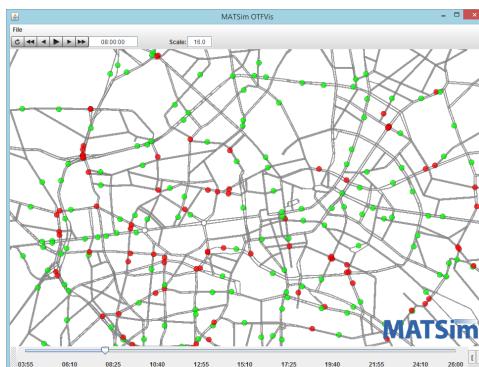
The website includes the latest guides for getting started with the tool, as well as detailed reference details for the many visualization types available in the tool.





# OTFVis

Author: David Strippgen



## 7.1 Basic Information

**Entry point to documentation:**

<https://github.com/matsim-org/matsim-libs/tree/master/contribs> → otfvis

**Invoking the module:**

<https://github.com/matsim-org/matsim-libs/tree/master/contribs> → otfvis → OTFVis class, RunOTFVis class

**Selected publications:**

Strippgen (2009)

## 7.2 Introduction

For most MATSim users, Via's (Chapter 5) free branch will be a good solution for their visualization needs. However, if project demands reach beyond the given (extensive but fixed) abilities of the Via free version, there is another—though not as stylish—option for MATSim output visualization, called OTFVis.

OTFVis is short for “On the Fly Visualizer”, and was designed to support visualization of live simulation runs with MATSim. Therefore, one purpose of the OTFVis is the debugging of MATSim (input) data. Nonetheless, playing prerecorded OTFVis Movie File, not to be confused with the “Musical Video Interactive” file usually abbreviated mvi (MVI) files created from MATSim events is another way to use

OTFVis. Generally speaking, OTFVis serves as an open-source counterpart to the possibilities Via gives the MATSim community. OTFVis is written in Java and available as source code to extend for different MATSim projects' special needs. Hence, it is possible and desirable to actually extend the OTFVis functionality, incorporating the user's own data sets and visualizations.

## 7.3 Quickstart

The easiest way to use OTFVis is as follows:

```
... main( ... ) {
    Config config = ConfigUtils.xxx(...) ;
    Scenario scenario = ScenarioUtils.loadScenario( config ) ;
    Controller controller = new Controller( scenario ) ;
    controller.addOverridingModule( new OTFVisLiveModule() ) ;
    // *****
    controller.run() ;
}
```

Config options for OTFVis are available via the syntax

```
... main( ... ) {
    Config config = ConfigUtils.xxx(...) ;
    OTFVisConfigGroup otfConfig = ConfigUtils.addOrGetModule(
        config, OTFVisConfigGroup.class );
    otfConfig.setXxx(...);
    ...
}
```

These options overlap with what can be set inside the OTFVis GUI, but it is often more convenient to set them once in Java instead of having to set them every time in the GUI.

The following text describes additional aspects of OTFVis. Please recognize, however, that these features are no longer supported by the core team.

## 7.4 More Ways of Using OTFVis

In this chapter, we show how to achieve simple things, like creating MVI-files from MATSim run events, how to play these MVI-files and how to use a MATSim config file to view/play an actual simulation with all data (e.g., agents' plans) attached. With the latter, it is also possible to examine the data "on the fly" by sending queries into the mobsim and visualizing the results.

### 7.4.1 MVI Files (unmaintained)

MVI files can be generated through OTFVis. Under the hood, these files consist of a few binary dumps of OTFVis data packed into a zip-file. This binary data is created by Java's own serialization capabilities. Unfortunately, this setup is not very change-resistant, making it advisable to regard MVI files as temporary cached versions of your event files. These MVI files can be re-created at any time from the event files. Still, as converting one into the other is a time-prone process, the MVI files are a handy tool for temporary storage and fast loading of your visualizations.

### 7.4.2 Starting OTFVis from the commandline (unmaintained)

OTFVis is a MATSim contribution. There is no actual stable release of the OTFVis package; so, to acquire a working version, a "nightly build" needs to be downloaded as shown in Section 22.3.6. There, one finds the latest `otfvis-version-SNAPSHOT-build.zip` file available for download. Unzip it to the place where

the `matsim.jar` already resides; do not forget to extract the `libs`-directories found in the respective zip files.

OTFVis demands substantial Random Access Memory (RAM) (depending on your simulation size/MVI file); to successfully launch the visualizer, a command line like

```
java -Xmx500m -cp matsim-XXX.jar:otfvis/otfvis-XXX.jar org.matsim.contrib.otfvis.OTFVis
```

(exchange “;” with “:” depending on the used Operating System (OS)) is a good starting point. This will open the dialog window shown in Figure 7.1, asking for one choice from four possible usages of OTFVis; these will be explained in the next section.

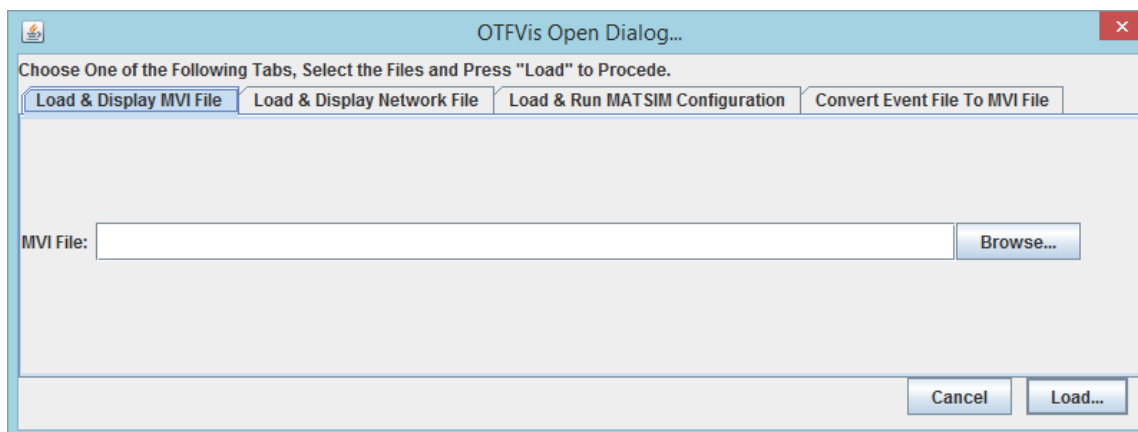


Figure 7.1: OTFVis Start Dialog.

### 7.4.3 Use Cases of OTFVis (unmaintained)

With the open dialog appearing after starting the vanilla OTFVis class, the following options appear, as shown in Figure 7.1:

1. opening a prerecorded MVI file,
2. opening a network file (for inspection),
3. opening a live run of a MATSim config file (rather memory intensive) or
4. converting an event file (plus a given network file) to a movie (MVI) file.

Each tab stands for an individual usage. To start a visualization, one chooses the appropriate tab, fills in the necessary data and finally proceeds by pressing the `Load...` button located in the bottom left corner of the window.

The next sections provide an overview of different ways to use OTFVis.

**Converting Event Files** Though the first option tab is the most used choice for OTFVis, the fourth, and last, option tab is a good starting point for exploring the visualizer; after having successfully run a MATSim simulation, there will typically be some event files at one’s disposal. With any of these event files and a given (matching) network file, a MVI file can be created. Four items: event, network and movie file names, as well as a time period, must be specified for this tab to execute. The last parameter is a time period, after which a new sample of the mobsim’s state is taken. This MVI-generation process might be time consuming. For smaller projects, it might be an option to display the outcome in the visualizer right away (by checking the box `Open mvi afterwards`). If the choice is to just convert the events to a MVI file, this can be opened with the first option tab of the visualizer’s start dialog at any time.

From the shell, this process can be started by giving the event file, network file and, optionally, the conversion period as input parameters.

**Network File Loading** The second option tab offers the opportunity to examine a network file (e.g., for errors). It will show a rendering of the given network and also, if so chosen in the preferences, the associated network link IDs for each link. This option might be helpful for debugging a freshly converted network, or inspecting specific regions and connections. Loading and interacting with a network file should be very fast.

The network file can also be given as the sole parameter to OTFVis with the shell command.

**Running a MATSim Configuration** The third, and most advanced, option for running OTFVis is an actual, live running mobsim, visualized in real time (actually much faster than real time; who has all day to watch tiny cars drive around?). This option includes the possibility of exploring the data set and issuing queries into the executing mobsim. These queries can display an agent’s day plan, show all links driven by agent’s crossing a particular link of interest, search for a particular link or node by ID, or answer any user-defined queries. We will see later in this chapter how to program a user’s own queries, but for the rest of this section we will detail OTFVis “offline” behavior.

It is also feasible to input the config file as a single parameter to OTFVis by starting it from the shell. OTFVis will make an educated guess whether the input is a config or a network file.

**Loading & Displaying an MVI File** If the first and default option tab is chosen, a MVI file is selected and shown as detailed in next section 7.4.4. This is the most common use case for OTFVis; the same results can be achieved by starting OTFVis from the shell with an MVI file as an argument.








#### 7.4.4 Viewing an MVI File (unmaintained)

An example is illustrated in Figure 7.2. On the top left of the application, one finds buttons for controlling the file playback. A short summary of the functionality is given in Table 7.1.

This buttonbar is followed by a text field where the desired time can be written for an instant jump. In an MVI file, one can jump forward and backward in time, whereas in the live simulation case, going back in time is omitted.

Another way of iterating through the animation is to grab the time slider at the bottom of the application and drag it. Opening and closing bracket symbols are located on the left side of the slider; by clicking them, one can set the start, or end, time of a time loop to the actual time step given, making it possible to restrict playback to a certain space of time.

Table 7.1: OTFVis Buttonbar.

Icon	Function
	Reset - set time to the start time
	Large step back
	Small step back
	Play
	Pause
	Small step forward
	Large step forward

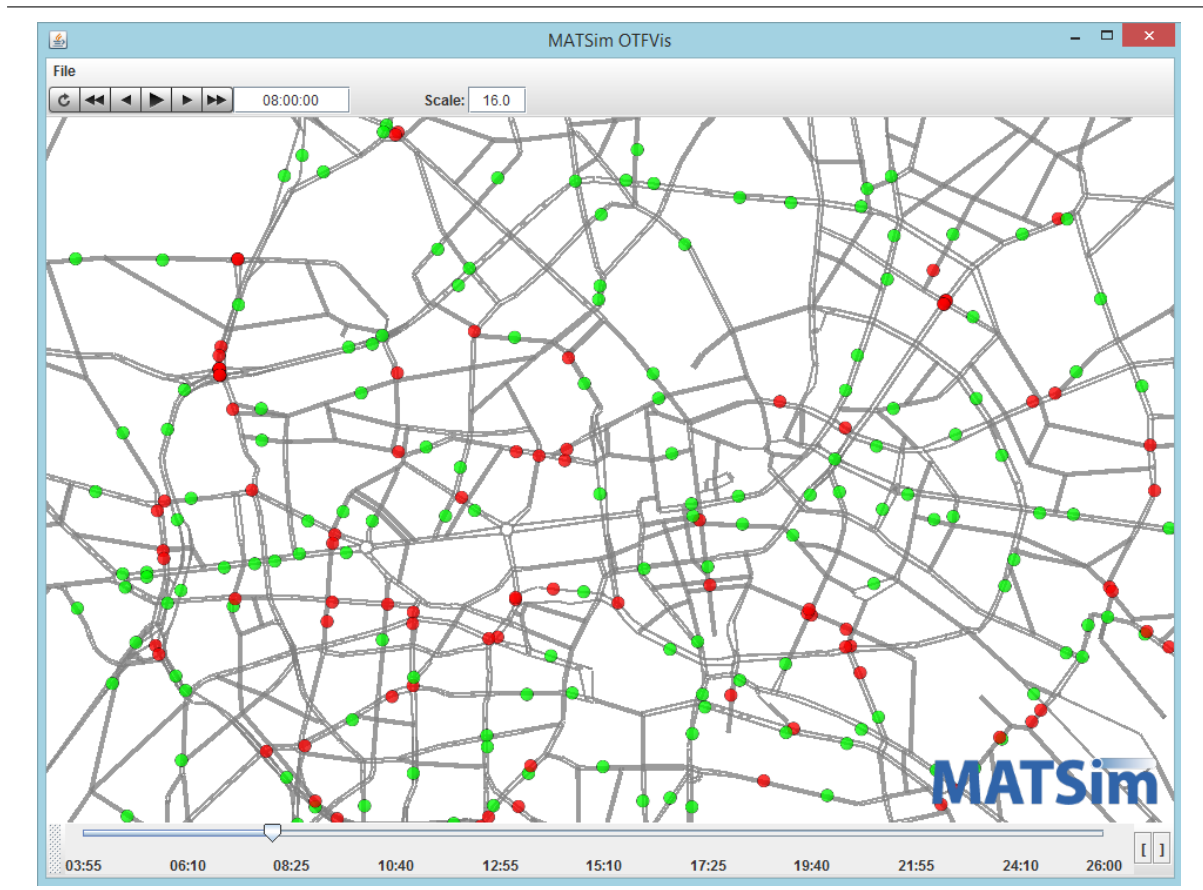


Figure 7.2: Displaying an MVI file.

#### 7.4.5 General Interaction with the Main Screen

Regardless which option for loading data was chosen, interaction with the main display area is the same.

**Right button drag:** Extend a rectangle for zooming into the view. Releasing the button will execute a zoom, so the chosen rectangle will best fit the screen.

**Middle-Mouse-drag:** Pan (translate) the screen.

**Right-Mouse-Click:** Show a context menu (for now only with the option to save the view settings).

#### 7.4.6 User Interaction in the Live Mobsim

When started as a live simulation, OTFVis will look different than Figure 7.3. First, the controls of the simulation's view flow are a restricted subset of those used in MVI playback. There is no way to reset or rewind the simulation. One can still take small or large steps forward. A new option is given by the synch checkbox, which determines whether the mobsim will stop for each frame the OTFVis renders, or run independently. Usually the un-synched version will proceed faster, as the OTFVis output is restricted to a default of about 30 frames/updates per second and a small mobsim's simulation speed will be a magnitude higher. The time-consuming generation of visualization data will also only be necessary for a small fraction of the simulation. Length of OTFVis pauses between frames can be configured in the preferences dialog.

Apart from the reduced control set, there is another UI element new to this OTFVis option. At the bottom of the screen, the scrubbar/time line element is replaced by a “query” bar. It is possible to code “queries” into the mobsim, answering questions about its inner state. As the simulation is actually happening, all information necessary to run it is available for output. This is a clear superset of information available in the event files and in the MVI files. This rich information infrastructure can be queried and visualized in many ways. In the next session, a query example is given.

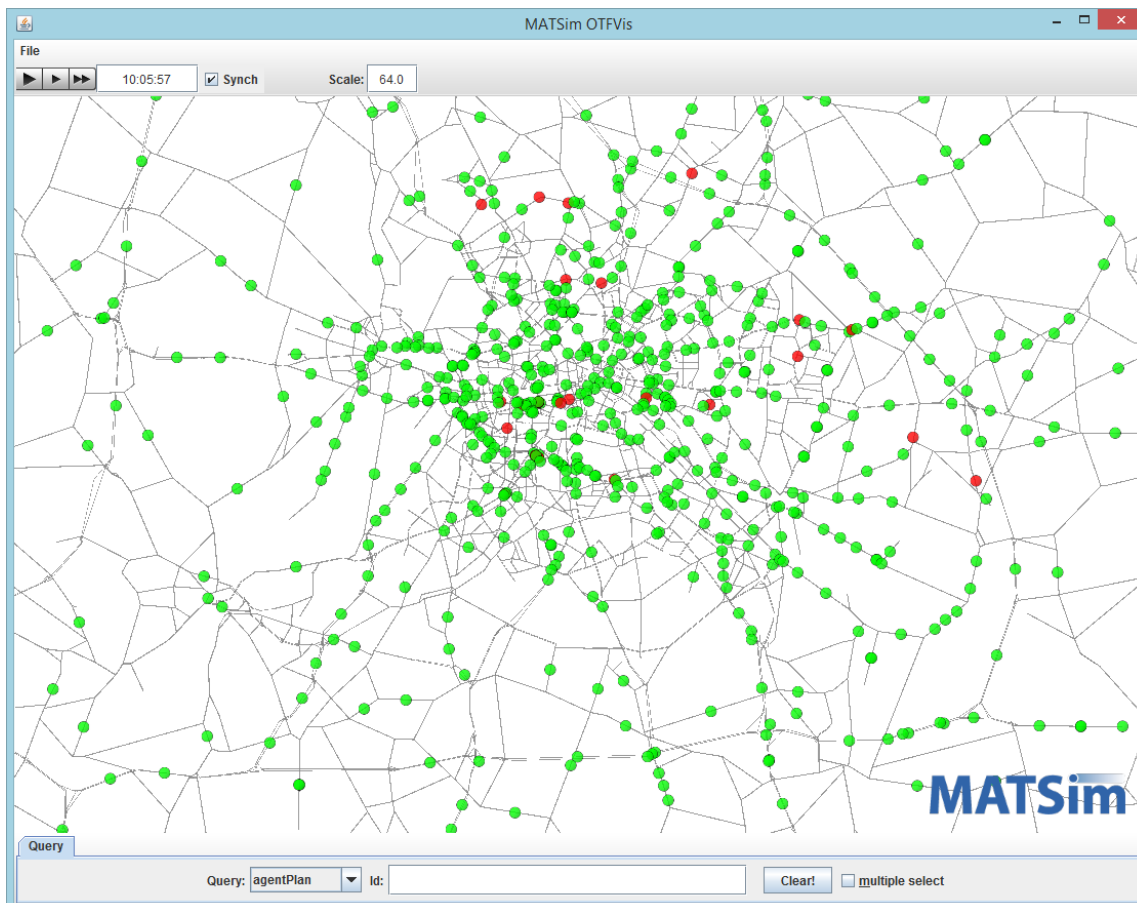


Figure 7.3: Live mode.

#### 7.4.7 Running a Query in OTFVis Real Time Data

From the dropdown box, one can choose the different query types. Often, additional input is necessary, either in the text field next to it or, more often, by clicking into the network. To give an example with agent query selected, a click onto any agent’s symbol will give a visualization of this particular agent’s day plan. This is shown in Figure 7.4. There are other pre-defined queries. These queries are rather project-oriented, so defining own queries will probably be necessary to make best use of this option. In the second part of this chapter, we will look into defining own queries.

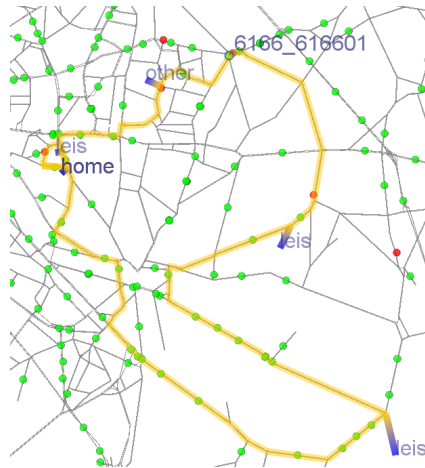


Figure 7.4: Queries.

## 7.5 Extending OTFVis

Because it is open source, the OTFVis is a good starting point for customizing mobsim run visualizations. OTFVis has been written in Java, but depends heavily on the Java OpenGL (JOGL) Java library. JOGL is a very thin layer within the OS hardware driver, meaning it will have OS-specific, native dependencies. These should be attended to by the maven-dependency management, but should still be kept in mind when developing for OTFVis. The displaying parts of OTFVis are based on Open Graphics Library (OpenGL). Therefore, it will be necessary to understand OpenGL to create new ways of displaying data. In the following sections, we examine how data is computed inside the OTFVis and how this can be extended.

### 7.5.1 Design Principles of OTFVis

The overall goal of OTFVis design was to have an easy-to-extend, fast visualizer capable of handling huge amounts of data. The specific design goals for the visualizer were:

- abstract data source (data collection) from data display (visualization),
- easy extension with own data types,
- capability for local simulation run on desktop computer,
- reduction of sent data to a minimum,
- visualization that connects to running simulation (on-the-fly),
- minimally-invasive format for existing MATSim code,
- enough speed for large scenarios,
- visualization that reads from post-mortem dump (MVI file), and
- use of hardware support for drawing.

MATSim runs can easily engage millions of agents traveling a network. To make a visualization of these large data sets feasible, two measures have been taken. A quad tree structure was implemented to ensure that only the smallest set of data necessary to display the visible sector of the network is transferred. The quad tree is a simple data structure to aggregate spatial data and retrieve parts of it efficiently for real time visualization. Apart from data structures, hardware is also used to speed up displaying the simulation. OpenGL is a platform-independent API for interfacing graphics hardware,

specifically the 3D acceleration chips implemented in every contemporary computer. With the aid of 3D graphics hardware, millions of agents can be displayed in real time. Other measures were taken to segregate data extraction from data visualization, like the reader/writer pairs presented in the next section.

### 7.5.2 Readers and Writers

OTFVis was designed to be minimally dependent on the mobsim used. Data formats applied within the mobsim should be abstracted from data used in OTFVis, meaning that any data passed to the visualizer will have to run through some stages of abstraction.

The first stage is a writer-reader pair, responsible for transferring a certain set of data to the OTFVis. The writer will understand the data format of the hosting mobsim and convert it to simple data types, like float or string values. A set of these writers, all using a joint byte buffer to aggregate the data, will be called after each mobsim step to accumulate data. This array of bytes is then sent to the visualizer, which, in the original design, could be run anywhere in your network.

For each writer, there has to be a sibling-reader class, responsible for reading back extracted data from the byte buffer. It is crucial to ensure that these pairs work synchronously. Most Writer/Reader-pairs are implemented in the same class, since having the source-code at the same place reduces errors in the synchronization.

Apparently, it can be necessary, or at least useful, to have different ways of visualizing data on the OTFVis front-end. Thus, actual readers are not responsible for the drawing of a certain data set. A third kind of class is responsible for that, the drawer classes.

### 7.5.3 Visualization of the Data

The reader objects in the quad tree will generate separate drawer objects for displaying “their” information and add these to another data structure, called SceneGraph, which is responsible for the actual drawing onto an OpenGL canvas. Displaying data in an interactive application will make re-draws of the display necessary for a variety of reasons: displaying menus, animations, zooming, panning and other user interactions. Not all of these changes introduce new data from the mobsim. Zooming into the network will not imply reading data from the mobsim; panning the view most certainly will. When no new data is needed, the scene graph is capable to handle all operations, no reader/writer class will be accessed and displaying is solely done with existing drawers. On the other hand, if new data is demanded, the scene graph will be “invalidated” (a term lent from the OpenGL community); thus, the graph will be dismissed and all relevant readers will be asked for new drawer objects representing the actual view. The scene graph is mainly a list of drawer objects; as an extra structuring unit, these drawers can be sent into different layers, to render them more effective.

### 7.5.4 Layers

To make sure that only data actually necessary for drawing the particular area visible in the viewport is sent, writers should minimize the data packets, so the quad tree can make a spatial data reduction. This seems somewhat in opposition to OpenGL or any graphics API. The API wants maximal data to be accumulated, to optimize output through the underlying hardware graphics pipeline. Think of an assembly line vs. a handcrafted item; whenever the flow of data is interrupted, the assembly line stalls and graphics performance derogates. To ease this issue, “layers” have been introduced to OTFVis. Any drawer (responsible for a bit of information) can be assigned to a layer and these layers will ultimately be summoned to draw the screen’s content. It is up to the layer to optimize the execution of the drawers when necessary. For example, a network layer might store all network info from the drawer in one array, or display a list to optimize drawing of the network; (often in OpenGL, it is advisable to



rather let the hardware decide what to draw. It might be faster to have all complete data residing in graphics hardware memory, rather than to transfer the reduced information set every frame). There are three layers predefined in OTFVis. The `networkLayer` contains the static street net, the `agentLayer` the actual dynamic agents and a third layer, the `miscellaneousLayer`, contains additional data.

### 7.5.5 Patching the Connections

In total, there are four basic elements involved in the visualization: writers, readers, drawers and layers. An additional class configures how the first two work together: `OTFConnectionManager`.

This class maps several routes for the information coming from the mobsim, building a chain of responsibility. Each data item starts at a link in our network. An `OTFDataWriter` object is responsible for extracting the desired data from the link and writing it into a `ByteBuffer`. Complementing this, an associated `OTFDataReader` is needed to retrieve data from the buffer. This item will also be responsible for adding a drawable item derived from the class `OTFGLAbstractDrawable` to the scene graph representing the actual screen content. The connection between these items is made by adding entries into the `OTFConnectionManager`, with calls to `OTFConnectionManager.connectLinkToWriter(OTFDataWriter)` and `OTFConnectionManager.connectLinkToWriter(OTFDataWriter, OTFDataReader)`, respectively.

Example (from the `OTFClientLive.java`):

```
conMan.connectLinkToWriter(OTFLinkAgentsHandler.Writer.class);
conMan.connectWriterToReader(OTFLinkAgentsHandler.Writer.class, OTFLinkAgentsHandler.class);
```

### 7.5.6 Sending the Data

The class `OTFLinkAgentsHandler` should give a good example of extracting, sending, receiving and displaying data in the OTFVis context. The method `invalidate` is called whenever the actual scene graph has been dismissed and needs to be rebuilt. In this case, a valid representation of the object's state should be added to the new scene graph. This also means that for drawing the actual scene, no additional reading will take place, unless there is a change in the visible data: then, this update is triggered.

### 7.5.7 Performance Considerations

When implementing new ways to visualize data, the following guidelines should be kept in mind.

If the data is spatially distributed over the whole network and is updated frequently, an `OTFDataWriter/Reader` pair should be considered. It will reduce data updating to times when the data is actually visible, not creating, transporting or drawing the data otherwise. If a fraction of the data needs to be transferred only once—because it is static over the time of the simulation—it can be sent with the `writeConstData()` method; otherwise using `writeDynData()` is advised. If the data is sparse and little information is transmitted or it has no discernible spatial cohesion, it might be simpler to just add it to the server quad tree as additional data with a call to `OTFServerQuadTree.addAdditionalElement()`.

### 7.5.8 Sending Live Data

Flow of data within OTFVis is almost always a one way affair, except for one important issue: sending queries into the simulation. In case of a live simulation run, visualized with help from the `OTFVisLiveClient` class, queries can be sent into the simulation. Again, the methods involved in this process are threefold; queries will be realized through an object derived from the abstract class `AbstractQuery`. Such an object initiates several methods that will be used as callback over the lifetime of the query.

First, a new query is sent to the server and the method `installQuery()` is called. In this method, all relevant parts (network, population, events) of the simulation run can be accessed and data can be collected. The visualizer framework will later repeatedly call the `result()` method, to retrieve an `OTFQueryResult` object. This has to implement a `draw()` method, to visualize the results in the given screen context. If the result indicates `isAlive()`, the `query()` method of the `AbstractQuery`-derived object will be called with each frame; otherwise, only once.

**Part III**

**Generation of MATSim**

**input**

## 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 Document Type Descriptions (DTDs). 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.

### 8.1 Coordinate Transformations in Java

Section 2.3.4.3 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.

## 8.2 Network Generation

### 8.2.1 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 is [RunPNetworkGenerator.java in matsim-code-examples](#).

### 8.2.2 From Other Sources

Networks can also be obtained from other sources. An example script of how to convert an EMME network to MATSim is [RunNetworkEmme2MatsimExample in matsim-code-examples](#). 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.

## 8.3 Initial Demand Generation

### 8.3.1 Simple Initial Demand

A simple script to generate a population with a single synthetic person with one initial is [RunPOnePersonPopulationGenerator in matsim-code-examples](#). A somewhat larger synthetic population is generated by [RunPPopulationGenerator in matsim-code-examples](#).

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.

### 8.3.2 Realistic Initial Demand

A script to illustrate the generation of a more realistic population and initial demand is [RunZPopulation-Generator in matsim-code-examples](#), generating a sample population from a census file and writing it to a file.

# Eqasim

**Author** (of this documentation): Kai Nagel

## 9.1 Basic Information

**Entry point to documentation:**

<https://eqasim.org/>.

**Invoking the module:**

**Selected publications:**

Hörl and Balac (2021).





**Part IV**

# **Additional core features**

## Additional MATSim Data Containers

Authors: Kai Nagel, Marcel Rieser, Andreas Horni

### 10.1 Attributable

MATSim operates on data types such as links, nodes, persons, or vehicles. Most 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. For these older approaches, see Sec. 21.4.1.

In XML, the `Attributable` approach looks as follows:

```
<person id="1">
  <attributes>
    <attribute name="age" class="java.lang.Integer">21</attribute>
  </attributes>
  <plan ...>
    ...
```

In this way, the age is attached to the person. Arbitrary attributes can be added in this way. They will be read in, are accessible in code, and are also written out again.

The usage in code is as follows: Data types implementing the `Attributable` interface allow a syntax of type

```
<object>.getAttributes().putAttribute( String key, DataType value ) ;
```

The content can be retrieved by

```
DataType value =
  (DataType) <dataType>.getAttributes().getAttribute( String key ) ;
```

It is important to do the cast into the correct type on the same line since `getAttribute(...)` actually knows the runtime type of the attribute, and thus generates a meaningful error message when the cast fails.

For a working example see [RunPersonAttributesExample in matsim-code-examples](#).

Arbitrary data types can be added as attributes. However, writers and readers only exist for basic data types such as numbers, strings, or enums. For data types that go beyond that, one needs to register specific `AttributeConverters`.

## 10.2 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.

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.

## 10.3 Subpopulations

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">
    ...
  </plan>
</person>
```

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 10), e.g.,

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

See [RunSubpopulationsExample](#) in [matsim-code-examples](#) for an example.

## 10.4 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 [counts100.xml in matsim-code-examples](#).

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 10.1) in a pop-up window.

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.

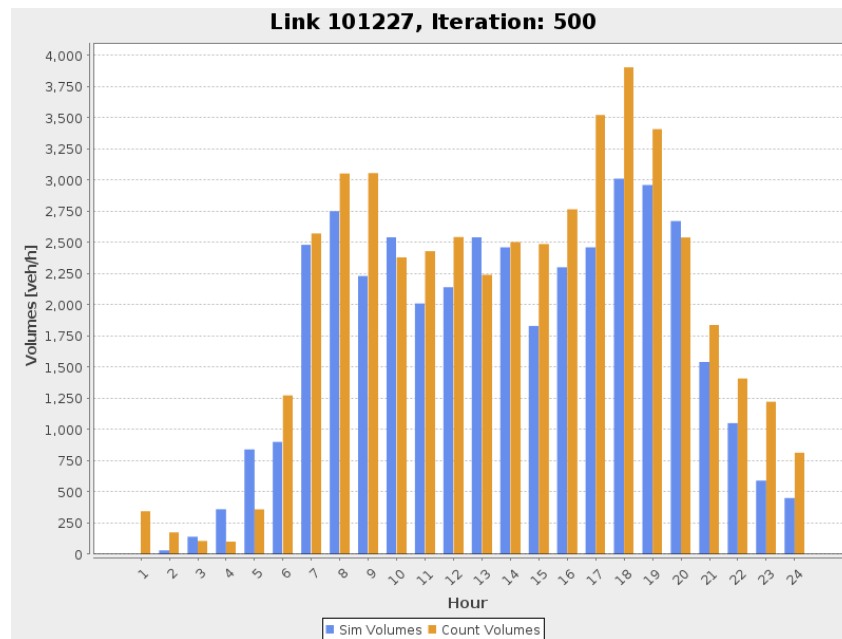


Figure 10.1: Example for a link volumes comparison between simulation and road count values.

## 10.5 Facilities

Facilities are an optional element of MATSim; some modules, such as the destination innovation module (Chapter 20), 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>
```

See [RunWithFacilitiesExample.java](#) in [matsim-code-examples](#) for an example.

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).

Loading a facilities file into MATSim by itself does not mean they will be used in the way one would expect. MATSim originally worked without facilities, and in an effort to make the initial data requirements minimal, facilities are still not needed to get MATSim up and running. However, if no facilities are

provided, MATSim will auto-generate them, with one facility per link, write them to a file, and use them. This has the effect that, if the facilities file is provided as input, it will be used.

For the time being, the core functionality of a facilities file is to contain the coordinate of the facility, which may be different than a coordinate derived from the network, and thus allow higher spatial resolution for activities. Any other functionality of the facilities, e.g., information about facility opening times depending on activity type, needs to be switched on by other means. Currently, this is only possible by using some class with a main method.

## 10.6 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 path to a file containing households (parameter inputFile).

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.

## 10.7 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.<sup>1</sup>

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

---

<sup>1</sup>There used to be an additional “useVehicles” config switch. Since release 0.8.x that switch is gone.

# Multi-modal 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. Cf. Figure 11.4.

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.<sup>1</sup>

<sup>1</sup>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.

The following sections describe current MATSim multi-modal capabilities.

## 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** In the past, the Java code used the term `ModeRoutingParams`. This has now been changed to `TeleportedModeParams`. However, the old syntax still works, and therefore may be encountered in user and example code. In the config file, this has always been named `teleportedMode...`



## 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.<sup>2</sup>

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).<sup>3</sup>

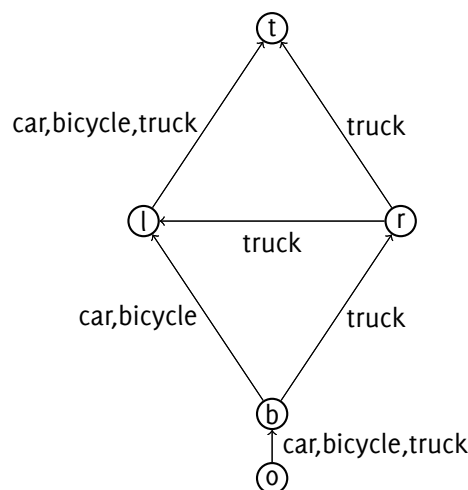


Figure 11.1: Multi-modal network example.

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.<sup>4</sup>

<sup>2</sup>Before release 11.0, it used to be the case that one also had to configure `travelTimeCalculator`, setting `separateModes` to true. This setting is now the default.

<sup>3</sup>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.

<sup>4</sup>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 these 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 12 and 18. They come with their own routers.

### 11.2.4 Other Routing Options

It is possible to register separate, possibly self-written routers for specific modes. This syntax is discussed in Section 21.6.9. 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 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 10), 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.

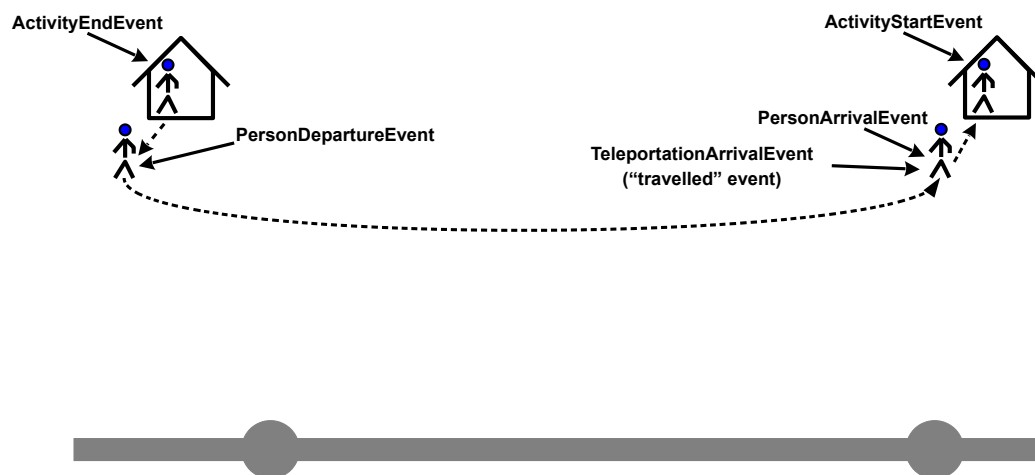


Figure 11.2: Teleported mode.

- 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,

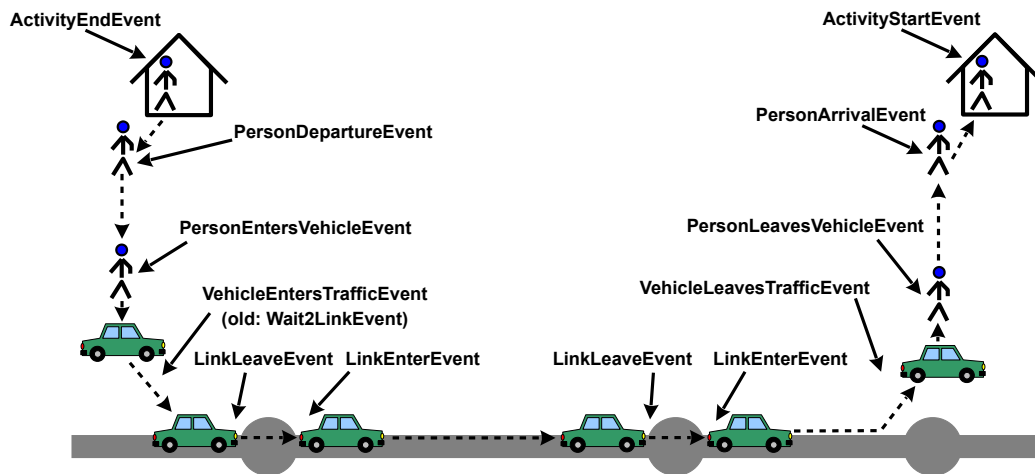


Figure 11.3: Network (= 'main') mode.

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.

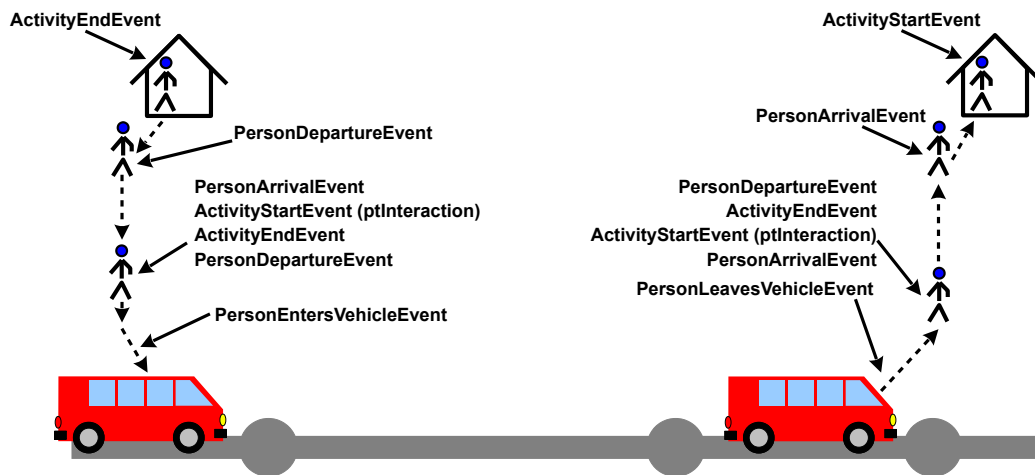


Figure 11.4: Passenger mode.

### 11.3.4 QSim Side: Departure Handlers

It is possible to register a separate departure handler for each mode; see Section 21.6.10 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 12).
- 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 18, 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 10.2.1 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 15), 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 10.7) 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 below), 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 10.7) needs to contain mode specific vehicle types, approximately as follows:

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

See [config-with-mode-vehicles.xml](#) in [matsim-code-examples](#) for a working example.

This approach can also be programmed as script-in-Java; see [RunMobsimWithMultipleModeVehicle-Example](#) in [matsim-code-examples](#).

## 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 First In, First Out (FIFO) 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 Legs, trips, and interaction activities

MATSim has a plans format that alternates elements of type Activity with elements of type Leg. This is modelled after the data model of GPS routing devices, which also use waypoints plus connections between them.<sup>5</sup>

This also means that inter-modal trips are encoded using only activities and legs:

```
<activity type="home" ... />
<leg mode="walk" .../>
<activity type="car interaction" .../>
<leg mode="car" .../>
<activity type="car interaction" .../>
<leg mode="walk" .../>
<activity type="work" ... />
```

“Interaction” activities (in the Java code called “stage” activities) are inserted between legs. In consequence, there are now interaction/stage activities, and “true” activities.<sup>6</sup>

<sup>5</sup>Much of MATSim is actually able to also deal with non-alternating sequences of activities and legs. However, at some point we decided that staying with the strictly alternating concept actually makes sense. In consequence, only strictly alternating plans are syntactically correct.

<sup>6</sup>The label for interaction/stage activities used to be defineable in MATSim, but has now been standardized.

There is also functionality to extract trips: `TripStructureUtils.getTrips(...)`. It does that by going from a “true” activity to the next, and gobbling up everything that is in between.

However, trips are not native objects in MATSim. The reason for this design decision was that much functionality of MATSim worked with alternating activities and legs, and it continued to work with interaction/stage activities. Another argument is that trips may actually have multiple hierarchical levels. For example, one might go from home to the airport, then fly to some destination, etc., but the (sub-)trip to the airport might by itself be composed of, say, walk and public transport legs. In consequence, a hierarchical format allowing multiple levels would have been needed. Overall, it was decided that all of these changes would require too much effort, with unclear gains given that it was possible to retrofit the existing “flat” format.

A consequence of this decision is that there is no data item to which one can attach properties of a trip. This had become most urgent with something called a “routing mode”, which memorizes which router was used to route a trip – for example, a pure walk trip can result both from a pure walk router and from a public transport router. The current convention is to attach such information to all legs of a trip, and to warn or abort if that information has become inconsistent between the legs of a trip.

## 11.7 Access/egress routing

The public transport (pt) router has always inserted access/egress legs to/from pt stops. This is now also possible – and strongly recommended – for all other modes:

```
<module name="planscalcroute" >
  <param name="accessEgressType" value="accessEgressModeToLink" />
  ...
</module>
```

or

```
config.plansCalcRoute().setAccessEgressType( AccessEgressType.accessEgressModeToLink );
```

This is particularly important when using multi-modal networks. Without access/egress routing, it may then happen that the desired mode (e.g. bicycle) is not allowed on the link to which the facility (or the activity) is connected. For example, if the facility/activity is directly connected to a motorway.

Access/egress routing in this case will search for the nearest link with the desired mode, and then create an access/egress leg to/from it.

A special case is if one wants to route the walk mode on the network. Since then that same walk mode cannot be used for access/egress to/from the walk network, in this case MATSim introduces an additional mode called `non_network_walk`. It will then demand teleportation parameters for this “bushwhacking” mode.

## 11.8 Other

The simulation is able to handle time-variant networks (Lämmel et al., 2010, see Section 10.2), 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 ??.



# Modeling Public Transport with MATSim

Author: Marcel Rieser

## 12.1 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)

## 12.2 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.

## 12.3 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 12.4.1 and Section 12.4.2, 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.

## 12.4 File formats

### 12.4.1 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 Public Transport (PT) 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"/>
    <passengerCarEquivalent pce="3.0"/>
  </vehicleType>
  <vehicle id="tr_1" type="1"/>
  <vehicle id="tr_2" type="1"/>
</vehicleDefinitions>
```

### 12.4.2 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 under 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 12.1;

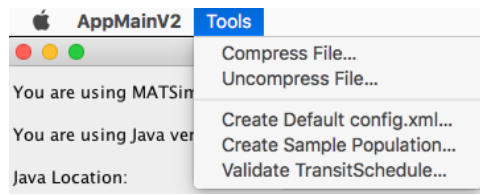


Figure 12.1: MATSim GUI tools.

alternatively, it can be run from the command line.<sup>1</sup> 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.

### 12.4.3 Events for a public transit trip

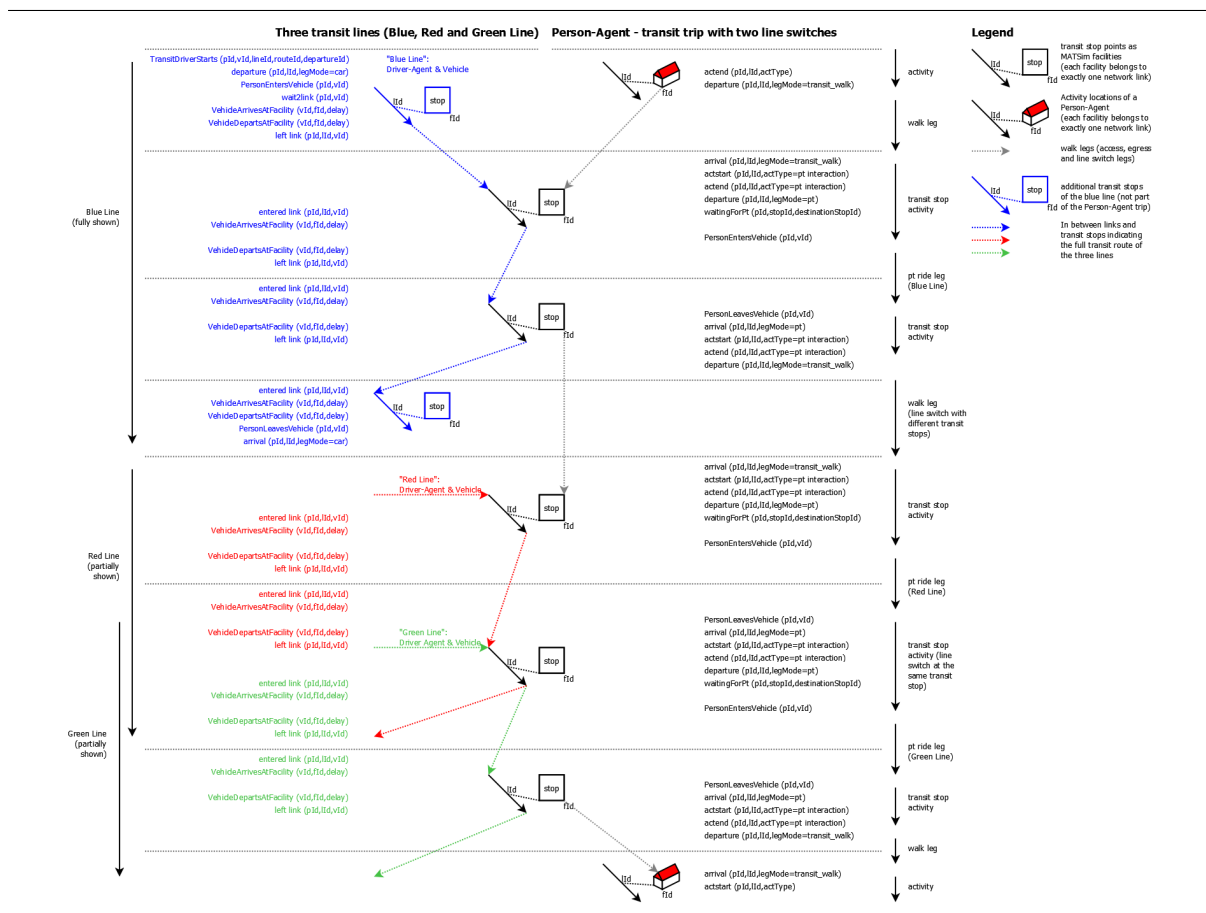


Figure 12.2: Events for a complex public transit trip.

<sup>1</sup>From the command line:

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

## 12.5 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.<sup>2</sup> More information can currently (May'21) be found here: <https://github.com/SchweizerischeBundesbahnen/matsim-sbb-extensions#swissRailRaptor>.

## 12.6 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 12.4.1) 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, occurring 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 Passenger Car Unit (PCU) 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<sup>3</sup> and in the QA.<sup>4</sup>

## 12.7 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 modeled: In MATSim, only one direction can go through; for the other, the

<sup>2</sup>If needed, the previous (much slower) transit router can be configured from the config by

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

<sup>3</sup><https://github.com/matsim-org/matsim-libs/pull/55>

<sup>4</sup><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, monetary-DistanceRate, 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. General Transit Feed Specification (GTFS) feeds can contain fare information, but more often than not those fields 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.<sup>5</sup>

---

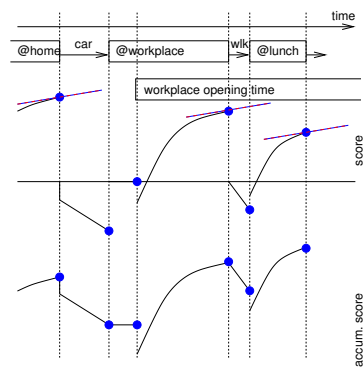
<sup>5</sup><https://gitlab.ethz.ch/ivt-vpl/populations/ch-pt-utils>,  
<https://doi.org/10.3929/ethz-b-000342816>





## A Closer Look at Scoring

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



### 10.1 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 22.6 and 22.6. 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 Artificial Intelligence (AI), 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 a large number of parameters to specify the scoring function. One can also add to the scoring function, or replace it.

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 10.2). This scoring function was, at times, modified, extended, or replaced for specific investigations (Section 10.6). Readily applicable estimates for a full-day utility function are not yet available, as discussed in Section ???.

## 10.2 The Current Charypar-Nagel Utility Function

### 10.2.1 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.

### 10.2.2 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  $S_{trav,mode(q)}$ :

$$S_{plan} = \sum_{q=0}^{N-1} S_{act,q} + \sum_{q=0}^{N-1} S_{trav,mode(q)} \quad (10.1)$$

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.

### 10.2.3 Activities

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

$$S_{act,q} = S_{dur,q} + S_{wait,q} + S_{late.ar,q} + S_{early.dp,q} + S_{short.dur,q} \cdot \quad (10.2)$$

The individual contributions are defined as follows:

**Performing an activity** The expression

$$S_{dur,q} = \beta_{perf} \cdot t_{typ,q} \cdot \ln(t_{dur,q}/t_{0,q}) \quad (10.3)$$

is the utility of performing activity  $q$ .  $t_{dur,q}$  is the performed activity duration,  $\beta_{perf}$  is related to the marginal utility of activity duration (or marginal utility of time as a resource, the same for all activities; see Section 10.2.7), 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 (10.3) as

$$= \beta_{perf} \cdot t_{typ,q} \cdot \ln(t_{dur,q}/t_{typ,q}) + \beta_{perf} \cdot t_{typ,q} \cdot \frac{1}{prio_q} \cdot \quad (10.4)$$

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  $\beta_{perf}$  at their typical durations.

Fig. 10.1 shows how this functional form reacts to changes of  $t_{typ}$ ,  $prio_q$ , or  $\beta_{perf}$ .

**$prio_q$  should be left at its default value of one** since other settings have never been systematically explored.

**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;  $\beta_{wait}$  is the so-called *direct* (see Section 10.2.7) marginal utility of time spent waiting; and  $t_{wait,q}$  is the waiting time. **We recommend leaving  $\beta_{wait}$  at its default value of zero**, except if good data about this effect is available, and the concept of the opportunity cost of time is well understood. Also see Section 10.2.8.

**Late arrival** The expression

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

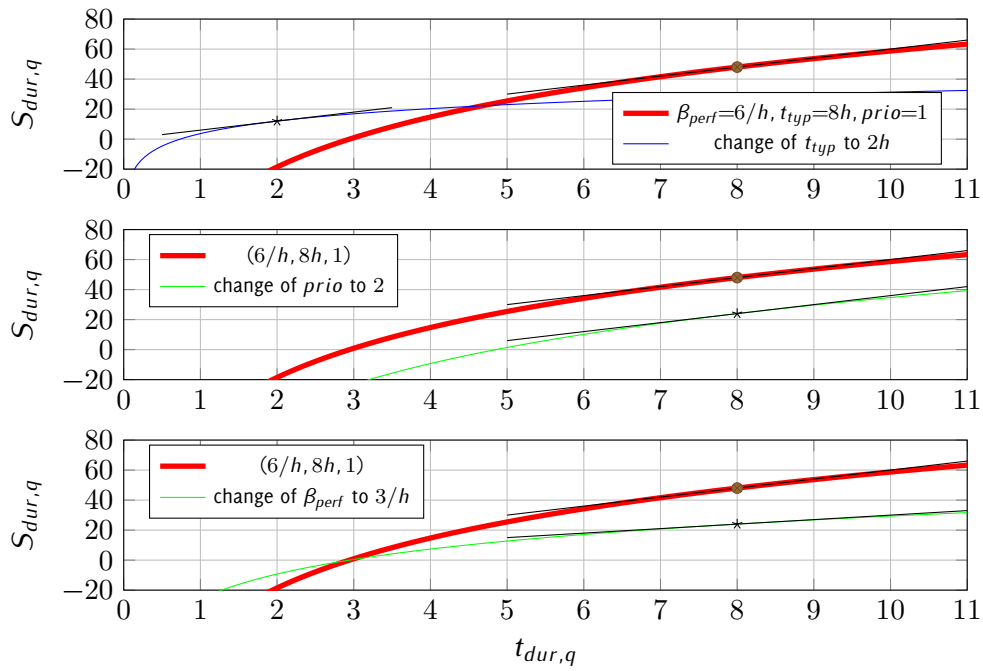


Figure 10.1: Reward for performing an activity when typicalDurationScoreComputation is set to relative. TOP: Effect of changing the typical duration. MIDDLE: Effect of changing *prio*. BOTTOM: Effect of changing  $\beta_{perf}$ . Note how, in the TOP plot, the score at the typical duration is proportional 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.

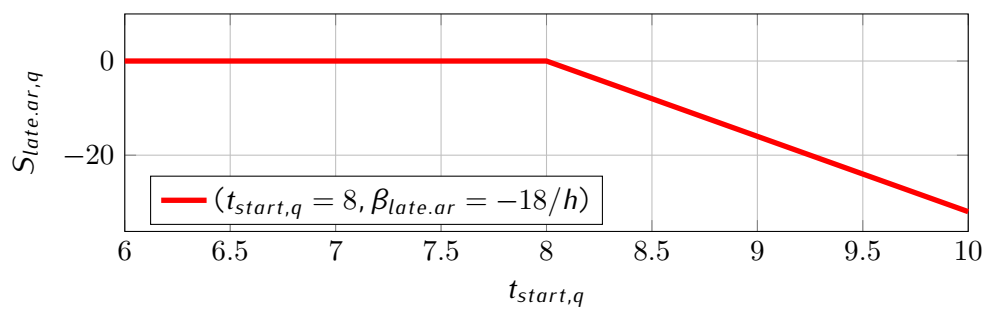


Figure 10.2: Penalty for arriving late.

**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 recommend leaving  $\beta_{early.dp}$  at its default value of zero, except if 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 recommend leaving  $\beta_{short.dur}$  at its default value of zero, except if 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.

## 10.2.4 Travel

Travel disutility for a leg  $q$  is given as

$$S_{trav,q} = C_{mode(q)} + \beta_{trav,mode(q)} \cdot t_{trav,q} + \beta_m \cdot \Delta m_q + (\beta_{d,mode(q)} + \beta_m \cdot \gamma_{d,mode(q)}) \cdot d_{trav,q} + \beta_{transfer} \cdot x_{transfer,q} \quad (10.5)$$

where:

- $C_{mode(q)}$  is a mode-specific constant.
- $\beta_{trav,mode(q)}$  is the *direct* (see Section 10.2.7) 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 10.2.7).
- $t_{trav,q}$  is the travel time between activity locations  $q$  and  $q + 1$ .
- $\beta_m$  is the marginal utility of money (normally positive).
- $\Delta m_q$  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 (10.5) is the direct utility contribution of travel; see Section 10.2.7 for the the full indirect utility as well as the relation to the Value of Travel Time Savings (VTTS), 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.

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  $Eu/m$ .

### 10.2.5 Illustration

Figure 10.3 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.

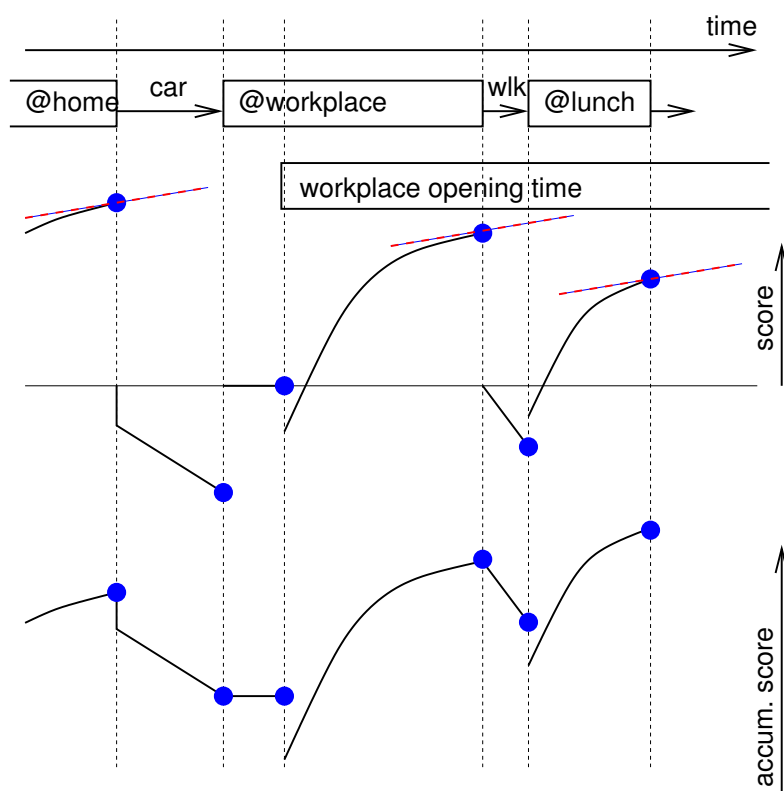


Figure 10.3: Illustration of the scoring function. TOP: Individual contributions of activities and legs. BOTTOM: Score accumulation over a day.

### 10.2.6 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 10.2.1, 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 leads to a different result, because of the nonlinear (logarithmic) form of the utility of performing. For example,  $\ln(1) + \ln(7) = \ln(7) \neq \ln(1 + 7) = \ln(8)$ .

### 10.2.7 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  $\Delta t_{trav}$ , 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<sup>1</sup>

$$mUTTS = -\frac{\partial}{\partial t_{trav}} S_{trav} + \frac{\partial}{\partial t_{dur}} S_{dur} = -\beta_{trav} + \beta_{perf} \cdot \frac{t_{typ,q}}{t_{dur,q}}, \quad (10.6)$$

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

<sup>1</sup>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  $\beta_{trav}$  that denotes how much better, or worse, it is to spend that time traveling, rather than “doing nothing”.<sup>2</sup> To differentiate  $\beta_{trav}$  from the indirect effect, it is sometimes called **direct marginal utility** of time spent traveling.

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

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

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

At the typical duration of an activity, this is

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

### 10.2.8 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,  $\beta_{wait}$ , 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  $\beta_{wait}$  separately from  $\beta_{perf}$ , **we recommend leaving  $\beta_{wait}$  at zero.**

**Arriving Late** Arriving late incurs a marginal utility of  $\beta_{late}$ , 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, (cf. Section ??).

**Vickrey Parameters** As a result, the Vickrey parameters of  $\alpha$  (marginal penalty for arriving early),  $\beta$  (marginal penalty for traveling) and  $\gamma$  (marginal penalty for arriving late), as defined by Arnott et al.

<sup>2</sup>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:

$$\begin{aligned} -\beta_{wait} + \beta_{perf} \cdot \frac{t_{typ,q}}{t_{dur,q}} &= \alpha \\ -\beta_{trav} + \beta_{perf} \cdot \frac{t_{typ,q}}{t_{dur,q}} &= \beta \\ -\beta_{late} &= \gamma \end{aligned} \quad (10.7)$$

## 10.3 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.

### 10.3.1 Negative Durations

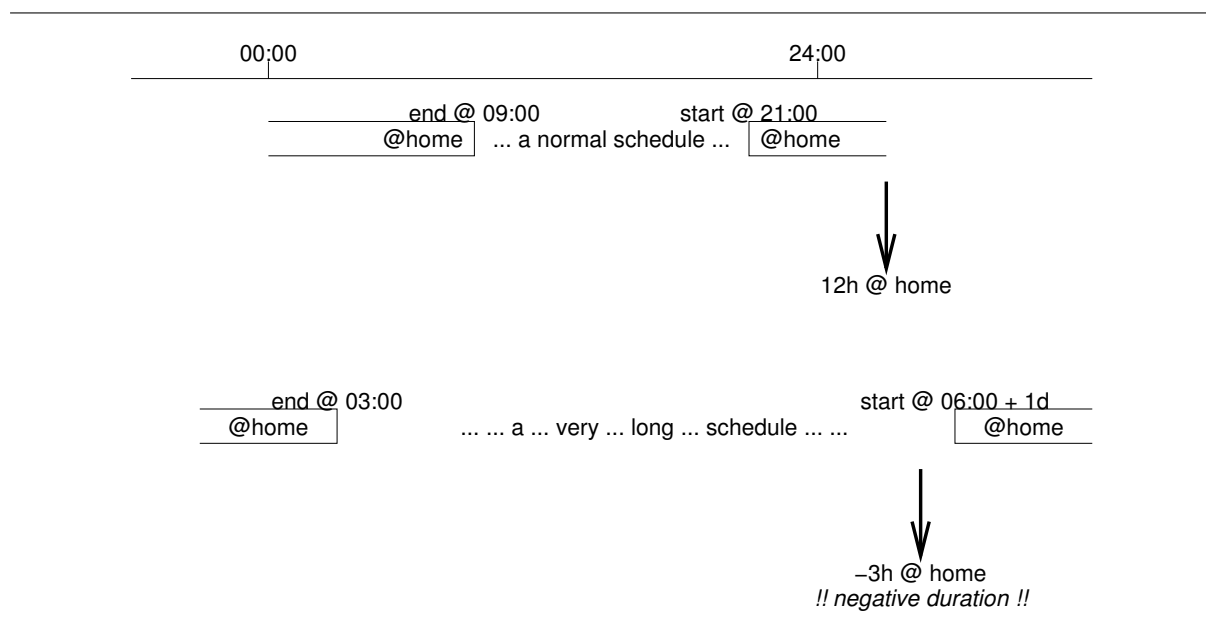


Figure 10.4: Illustration of wrap-around scoring. TOP: Normal situation. BOTTOM: Situation where the final activity of the plan starts at a later time of day than when the first activity ended, resulting in a negative duration of the wrap-around (= overnight) activity..

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 10.2.6). 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 10.4). 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 10.5).<sup>3</sup>

<sup>3</sup>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

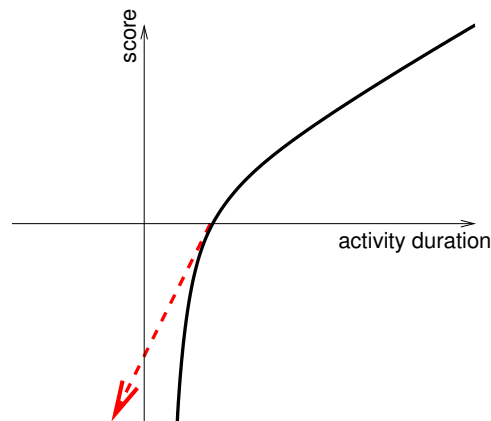


Figure 10.5: Extending the slope when the utility function crosses the zero line to negative durations.

**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.

### 10.3.2 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

$$S^k = \alpha S + (1 - \alpha) S^{k-1}, \quad (10.8)$$

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  $\alpha$  is a "learning" or "blending" parameter. The default value of  $\alpha$  is one; it can be configured by the line

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

in the config file.

Non-executed plans just keep their score.

### 10.3.3 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 (10.8) will not achieve that; it just dampens the fluctuations. A well-known approach to force convergence to the expectation value is MSA:

$$S^m = \frac{1}{m} S + \frac{m-1}{m} S^{m-1}. \quad (10.9)$$

This resembles Equation (10.8), with the important difference that the fixed blending parameter  $\alpha$  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.

## 10.4 Typical Scoring Function Parameters and their Calibration

The current MATSim default values are

$$\begin{aligned}
 \beta_m &= 1 \text{ utils/monetaryunit} \\
 \beta_{perf} &= 6 \text{ utils/h} \\
 \beta_{trav,mode(q)} &= -6 \text{ utils/h} \\
 \beta_{wait} &= 0 \text{ utils/h} \\
 \beta_{short.dur} &= 0 \text{ utils/h} \\
 \beta_{late.ar} &= -18 \text{ utils/h} \\
 \beta_{early.dp} &= 0 \text{ utils/h.}
 \end{aligned}
 \tag{10.10}$$

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. In many of the systems that we model, traveling by car does not seem to be less convenient than “doing nothing”. Thus, the *direct* marginal utility of traveling by car,  $\beta_{trav,car}$ , 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:<sup>4</sup>

1. Set  $\beta_m \equiv \text{marginalUtilityOfMoney}$  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. (This is the default.)  
This is normally a positive value, since having more money normally increases utility.
2. Set  $\beta_{perf} \equiv \text{performing}$  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. (This is the default.)  
This is normally a positive value, since performing an activity for more time normally increases utility.
3. Set  $\beta_{trav,car} \equiv \text{marginalUtilityOfTraveling...}$  to 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  $\beta_{perf}$ .* See Section 10.2.7.
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

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

then

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

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

<sup>4</sup>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  $C_{mode} \equiv \text{constant}$  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 12) goes beyond what can be provided here.

## 10.5 Helper functions

### 10.5.1 Experienced plans

There is the config option

```
<module name="scoring" >
  <param name="writeExperiencedPlans" value="true" />
  ...
```

It will write, in those iterations when regular plans files are written, also files `*experienced_plans.xml.gz` and `*experienced_plans_scores.txt.gz`. The first writes the actually experienced plans (rather than the “planned” plan); the second writes the individual contributions of activities and legs to the overall score.<sup>5</sup>

## 10.6 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 20), 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 ??).

---

<sup>5</sup>It is, unfortunately, not clear if this will write *all* contributions, or only those from activities and legs. Contributions that go beyond those from activities and legs are, for example “person money events” and “person score events”.

## 10.7 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 (10.3) 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. 10.6 illustrates the uniform setting. Section ?? discusses further alternatives.

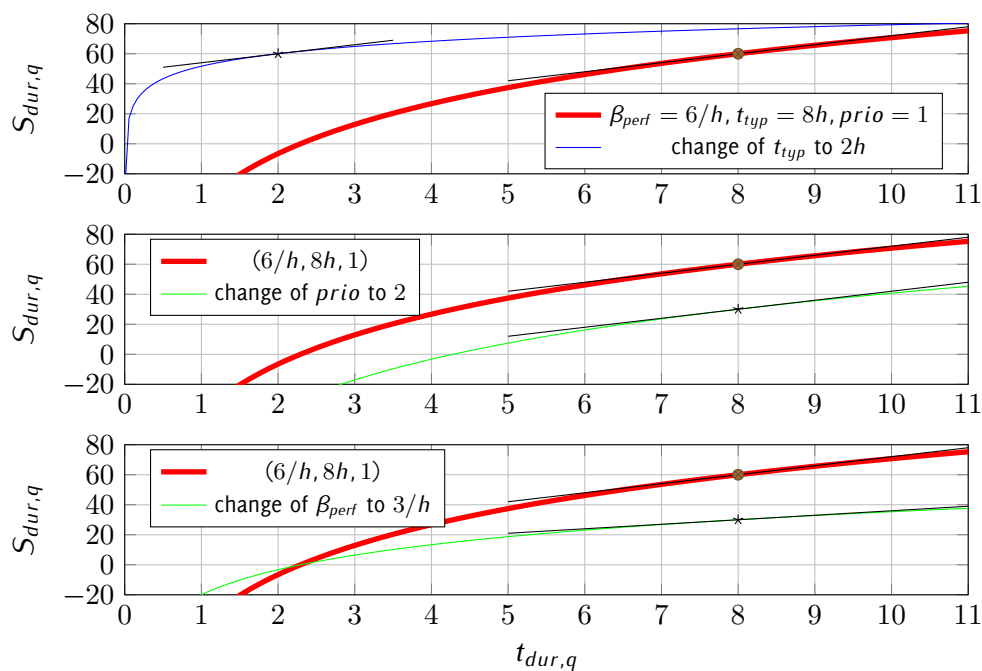


Figure 10.6: Reward for performing an activity when `typicalDurationScoreComputation` is set to uniform. TOP: Effect of changing the typical duration. MIDDLE: Effect of changing  $prio$ . BOTTOM: Effect of changing  $\beta_{perf}$ . Note how in the top plot the score at the typical duration just moves to the left, but not down, other than in Fig. 10.1.



**Part V**  
**Analysis**

## CSV outputs

Author: Billy Charlton

```
#EPSG:31468 (this is a CSV comment)
departureTime ,personId ,vehicleId ,fromLinkId ,from_x ,from_y ,to_x ,to_y ,travelTime
16.0 , "link-243466701" , "drt596" , "51717" , 4589322.86 , 5821008.32 , 4590199.68 , 5822078.39 , 103.0
33.0 , "link-419305801" , "drt716" , "136685" , 4592171.06 , 5813214.2 , 4593371.04 , 5811150.86 , 286.0
87.0 , "drt-223775601" , "drt1077" , "101393" , 4600723.1 , 5823415.31 , 4597578.72 , 5824512.74 , 342.0
...
```

Many standard MATSim outputs are in the CSV text file format. These output files are described in this section, along with some best practices for using the CSV format with MATSim .

### 11.1 MATSim standard CSV outputs

The following default outputs are created by MATSim, either in raw text form or compressed using gzip. Note the column delimiters used for each file, as they are inconsistent.

**output\_persons.csv.gz** - (semicolon-separated format) Statistics for every person in the simulation. Each row provides demographic data including age, sex, income, car availability; as well as simulation outputs such as executed score and initial activity coordinates. Depending on the simulation, other columns may also be present.

**output\_legs.csv.gz** - (semicolon-separated format) Statistics for every leg in the simulation. Each row includes the person ID, trip ID, departure/travel/wait times, leg distance, mode, starting and ending links and coordinates, and transit details for transit legs.

**output\_trips.csv.gz** - (semicolon-separated format) Statistics for every trip in the simulation. Each row includes the person ID, trip ID and number, departure/travel/wait times, traveled and euclidean distance, mode information, start and end activity types, starting and ending facilities, starting and ending links and coordinates, and transit information for transit trips. This file is the primary input for many of the post-processing functions in the MATSim R library described in section 13.

**\*modestats.txt, scorestats.txt, stopwatch.txt, traveldistancestats.txt** - (tab-separated format) These small files provide top-level summaries for mode, travel distances, and scores; one row per iteration.

### 11.2 Recommendations for using CSV in the MATSim context

CSV is a venerable text file format that has been in use for decades to facilitate data storage and exchange between programs. Due to its long pedigree, CSV enjoys extremely broad support for import



and output in just about every data-aware program. At its simplest, it provides a human-readable, compact data format that is editable anywhere. But it also suffers from some “gotcha” implementation details that are described below.

CSV is a row-based text format, with each row including columns of data separated by a delimiter character, usually but not always a comma. Hence the name, “comma-separated values”. The only information about the data in the file is in an optional header row which can provide labels for each column. There is no data type information, precision settings, or validation as part of the file format.

### 11.2.1 CSV file format variations: delimiters, header, quotes, and comments

**Delimiters.** The standard column delimiter of a true CSV file is the comma. However, in many European countries including Germany, the historical and typical representation of numbers inverts commas and periods from their international use as thousands and decimal separators. For this reason, some CSV files in central Europe use the semicolon as the separator instead of the comma. Furthermore the tab character (ASCII value 9) can be used instead of either comma or semicolon; this is often called “tab-separated” format or TSV. Tabs can make files a bit easier to read, but some text editors convert tabs to spaces so caution is required.

These different delimiters only cause problems if the user is unaware of the variations. Most CSV reading libraries can handle various delimiters and will either auto-sense it properly or it can be specified.

**Header line.** The first line of a CSV file may be an optional header which provides names of the columns of data that follow in all subsequent rows. Without a header row there is no way for a user to know what the file contains; this information must be provided out-of-stream somehow. For this reason, it is always recommended to use a header row when possible.

Over the years, many conventions have arisen regarding good column names: for example, geographic point data should include suffixes `_x`, `_y`, `_lat`, or `_lon` so that they are auto-sensed when imported into common GIS tools.

**Quotes.** String columns can be enclosed in “double quotes” for clarity although this is not required unless the string contains the delimiter field. If double quotes are used to enclose fields, then a double quote appearing *inside a field* must be escaped by preceding it with another double quote. For example: `“aaa”,“b”“bb”,“ccc”`

Many CSV files do not encode double quotes properly as above – be warned.

**Comments.** There is no official, standard way to include comments in a CSV file, but many libraries allow the use of the hash/pound character `#` in the first position to signify that the row is to be ignored by the parser. This is one way to provide information to the data consumer about what is included in the file.

For example, one could include the line `#EPSG:31468` as the first line of a CSV file to signify the coordinate reference system of point data that follows.

Not all CSV parsing libraries support this and it is nonstandard but common – so be warned.

### 11.2.2 Guidance on creating MATSim-standard CSV files

Given all of these variations, the core team of MATSim developers has some guidance on how to create well-formed, usable CSV files for successful data interchange:

- **Delimiter:** Use a comma or tab. Avoid using semicolon as the column delimiter.
- **Header:** Always include a header line that labels your data columns.

- **Column names:** Avoid spaces in names unless the column names will be directly used in visual outputs as labels. Always use suffixes `_x` and `_y`, or `_lat` and `_lon`, for coordinate data so that the data can be imported to other tools easily.
- **Comments:** Include a header `'#EPSG:12345'` signifying the coordinate reference system of any geographic point data, if that data is not in longitude/latitude format. Note this is not standard CSV, and may cause interoperability issues.

With these tips, using CSV with MATSim can be a consistent and pleasant experience.

# 12

## Python bindings

Author: Billy Charlton

```
import matsim
import pandas as pd
from collections import defaultdict
%matplotlib inline
# -----
# 1. NETWORK: Read a MATSim network:
net = matsim.read_network('output_network.xml.gz')
net.nodes
# Dataframe output:
#      x      y node_id
# 0 -20000.0    0.0      1
# 1 -15000.0    0.0      2
# 2  -865.0  5925.0      3
# ...
net.links
# Dataframe output:
#      length capacity freespeed ... link_id from_node to_node
# 0  10000.0  36000.0    27.78 ...      1         1         2
# 1  10000.0   3600.0    27.78 ...      2         2         3
# 2  10000.0   3600.0    27.78 ...      3         2         4
# ...
```

### 12.1 Basic Information

#### Entry point to documentation:

MATSim Python binding documentation is on the PyPi package library website at <https://pypi.org/project/matsim-tools/>.

#### Invoking the module:

Install using "pip install matsim-tools". Then "import matsim" from your Python code to access the API.

#### Selected publications:

-

## 12.2 Introduction

The beginnings of a Python API for accessing MATSim output files is now available. The feature set is still under development, so expect the API to change and features to be added in time.

The main goals of the Python API:

- Read MATSim outputs into Python in a way that works well with the pandas and geopandas libraries, for data analysis workflows.
- Initially support reading network, event, and plans files
- Support XML, JavaScript Object Notation (JSON), and Protobuf event file formats
- Initial support for writing MATSim files

## 12.3 Installation

Python 3.6 and above are supported.

Packages and install files are available on PyPi at <https://pypi.org/project/matsim-tools/> and also on Condaforge at <https://conda-forge.org/>.

Install from PyPi using "pip install matsim-tools". Alternatively the source code can be installed from <https://github.com/matsim-vsp/matsim-python-tools> if you do not wish to use a package manager.

## 12.4 Using the MATSim Python API

MATSim Python API is designed to either use the pandas data library directly, or work well with it.

The full API is still under development and is not duplicated here. See the PyPi documentation site linked above for API documentation and sample code. Some examples are reprinted here.

**Read a network.** This code loads a network file in as two pandas dataframes, one for nodes and one for links.

```
import matsim

net = matsim.read_network('output_network.xml.gz')

net.nodes
# Dataframe output:
#      x      y node_id
# 0 -20000.0  0.0      1
# 1 -15000.0  0.0      2
# 2  -865.0  5925.0     3
# ...

net.links
# Dataframe output:
#      length  capacity  freespeed  ...  link_id  from_node  to_node
# 0  10000.0  36000.0    27.78  ...      1           1           2
# 1  10000.0   3600.0    27.78  ...      2           2           3
# 2  10000.0   3600.0    27.78  ...      3           2           4
# ...
```

**Event processing.** MATSim event files do not convert easily to Pandas dataframes, because every event type has a different set of properties, while dataframes expect a well-defined set of columns. Depending on your use case, your options are to either (1) filter specific event types into separate dataframes (one for each type), or (2) collect the data you need into python dictionaries and/or lists which can be converted to dataframes at the end (or analyzed using regular Python).

Be warned that the event ORDER in MATSim event files is important, so separating event types into separate dataframes is often a bad idea. Option (2) above is a bit more work but very likely what you need to do.

Event example. More examples are available online.

```
# EVENTS: Stream through a MATSim event file.
# The event_reader returns a python generator function, which you can then
# loop over without loading the entire events file in memory.
#
# -----
# Example: Sum up all 'entered link' events to get link volumes.
# Supports both .xml.gz and protobuf .pb.gz event file formats!
# Only returns events of type 'entered link' and 'left link':
import matsim
import pandas as pd
from collections import defaultdict
events = matsim.event_reader('output_events.xml.gz', types='entered link,left link')

# defaultdict creates a blank dict entry on first reference; similar to {} but friendlier
link_counts = defaultdict(int)

for event in events:
    if event['type'] == 'entered link':
        link_counts[event['link']] += 1

# convert our link_counts dict to a pandas dataframe,
# with 'link_id' column as the index and 'count' column with value:
link_counts = pd.DataFrame.from_dict(
    link_counts, orient='index', columns=['count']
).rename_axis('link_id')
```

**Plan files.** Each plan is returned as a tuple with its owning person (for now).

Some hints on working with plan files:

- Use "selectedPlansOnly = True" to only return selected plans
- Always emits person, even if that person has no plans
- Every element can be iterated on to get its children (e.g. the plan's activities and legs)
- The name of the element is in its .tag (e.g. 'plan', 'leg', 'route', 'attributes')
- An element's attributes are accessed using .attrib['attrib-name']
- Use the element's .text field to get data outside of attributes (e.g. a route's list of links)

An example of reading a plan file:

```
plans = matsim.plan_reader('output_plans.xml.gz', selectedPlansOnly = True)

# Each plan is returned as a tuple with its owning person
# - The name of the element is in its .tag (e.g. 'plan', 'leg', 'route', 'attributes')
# - An element's attributes are accessed using .attrib['attrib-name']
# - Use the element's .text field to get data outside of attributes
#   (e.g. a route's list of links)
# - Every element can be iterated on to get its children
# - Emits person even if that person has no plans

for person, plan in plans:

    # do stuff with this plan, e.g.
    work_activities = filter(
        lambda e: e.tag == 'activity' and e.attrib['type'] == 'w',
        plan)

    print('person', person.attrib['id'], 'selected plan w/',
          len(list(work_activities)), 'work-act')

    activities.append(num_activities)
```

```
# person 1 selected plan w/ 2 work-act  
# person 10 selected plan w/ 1 work-act  
# person 100 selected plan w/ 1 work-a
```

Be sure to review the PyPi website <https://pypi.org/project/matsim-tools/> for additional documentation as the Python API matures.

## R bindings

**Author:** Billy Charlton



The R language and platform is well-suited to data analysis workflows. There is now a MATSim-R support package under development which should make it easier to work with MATSim files in the R environment.

### 13.1 Basic Information

**Entry point to documentation:**

Documentation lives at <https://vsp.berlin/matsim-r>.

**Invoking the module:**

To install from Github, in an R session run `install.packages("devtools")` followed by `devtools::install_github("matsim-vsp/matsim-r")`.

**Selected publications:**

-

### 13.2 Introduction

The beginnings of an R API for accessing MATSim output files is now available. The feature set is still under development, so expect the API to change and features to be added in time.

The main goals of the R API:

- Read MATSim outputs into R in a way that feels natural and works well for data analysis workflows.
- Support reading network, trips, and eventually events and plans files

## 13.3 Installation

This package is not yet available on the CRAN R package repository, as it is still changing rapidly. Until it is on CRAN, it can be installed directly from the main Github repository using the R “devtools” library.

```
# only need this if you don't have devtools already:
install.packages("devtools")

devtools::install_github("matsim-vsp/matsim-r")
```

## 13.4 Using the R API

The best way to get familiar with the R API is to open the online help directly from R-Studio, using the “??matsim” command and then opening the help index for the matsim package. Each of the API functions are documented inline and so the docs appear automatically in the R help system. Documentation is also online at <https://vsp.berlin/matsim-r>.

Currently, MATSim network files and standard output\_trips tables can be read into R as data frames. Then, a multitude of functions exist for creating plots and SimWrapper dashboards.

Note, the MATSim R library uses the R “tidyverse” internally; if you do not use the tidyverse, just know that tibbles and dataframes are interchangeable and have the same internal format.

Some example functions:

- **loadNetwork** - Loads a MATSim XML network file, creating a nodes tibble and a links tibble. Any node and link attribute records in the network are stored as additional columns in the respective node and link tibbles.
- **readTripsTable** - Loads a MATSim output\_trips CSV from file or gzip archive, creating a tibble with columns as in the CSV file
- **transformToSf** - Transforms trips table from tibble to an “SF” table with attribute features and geometry feature, for use with the SF geometry library
- **prepareSimwrapperDashboardFromFolder** - Creates inputs files for a SimWrapper interactive dashboard for the given output folder, based on the output\_trips table. After creating the files, use the SimWrapper website to view the dashboard
- **plotMapWithTrips, plotModalShiftBar, plot...** - Many plot... functions take the trip table as input and produce a specific plot of the data

The API continues to evolve so it is important to check the installed documentation to see what features are available.



## The “Analysis” Contribution

Author: Kai Nagel

### 14.1 Basic Information

**Entry point to documentation:**

<https://github.com/matsim-org/matsim-libs/tree/master/contribs> → analysis

**Invoking the module:**

No standard invocation. See the <https://github.com/matsim-org/matsim-libs/tree/master/contribs> → analysis RunKNEventsAnalyzer class for intuition.

**Selected publications:**

-

### 14.2 Summary

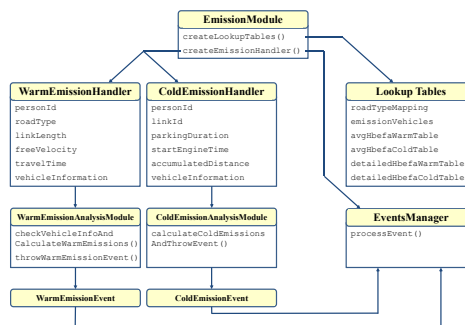
This contribution collects various analysis tools for MATSim output.

One important reason for having this in a contribution rather than in a playground is the Maven layout of the repository: Contributions can use material from other contributions, but not from the playgrounds. In consequence, analysis tools that are needed in a contribution need to be in a contribution themselves. The analysis contribution is a possible place where to put them.



# Emissions

Author: Benjamin Kickhöfer



## 15.1 Basic Information

Entry point to documentation:

<https://github.com/matsim-org/matsim-libs/tree/master/contribs> → emissions

Invoking the module:

<https://github.com/matsim-org/matsim-libs/tree/master/contribs> → emissions → RunEmissionToolOnlineExample class, RunEmissionToolOfflineExample class

Selected publications:

Hülsmann et al. (2011); Kickhöfer et al. (2013); Kickhöfer and Nagel (2011, 2016); Hülsmann et al. (2013); Kickhöfer (2014); Kickhöfer and Kern (2015)

## 15.2 Introduction

This chapter presents the emission modeling tool developed and tested by Hülsmann et al. (2011) and further improved by Kickhöfer et al. (2013). The text in this chapter is a slightly updated version of the emission modeling tool description in Kickhöfer (2014). The tool calculates warm and cold-start exhaust emissions for private cars and freight vehicles by linking MATSim simulation output to the detailed “Handbook on Emission Factors for Road Transport (HBEFA)” database, available for many European countries.

The chapter is structured as follows: Section 15.3 reviews literature for other attempts to model transport-related emissions. Section 15.4 presents an overview of the “Emission Modeling Tool, see Chapter 15 (EMT)” and Section 15.5 shows how the tool is embedded in MATSim’s software structure.

## 15.3 Integrated Approaches for Modeling Transport and Emissions

Over the last two decades, the modeling of transport-related environmental externalities has received increasing attention in transportation science. The following paragraphs briefly present some recent work in the exhaust emission modeling area; additionally, they highlight differences to the EMT, which will then be described in subsequent sections.

Creutzig and He (2009) and Michiels et al. (2012) use very aggregated figures to estimate air pollution in Beijing and Belgium, respectively. Neither approach mentions any particular underlying transport model. It seems that transport related emissions are based on aggregated origin-destination matrices or aggregated demand functions. These two studies are on a very different level of aggregation than the EMT, and a comparison does not seem constructive.

Beckx et al. (2009) use a sophisticated activity-based model to simulate activity schedules for roughly 30 % of all households in the Netherlands. Traffic assignment for passenger cars is performed by using an aggregated “all-or-nothing” assignment approach, resulting in hourly aggregated traffic flows on the network. Based on the average speed for a trip, the MIMOSA model then calculates emission and fuel consumption rates, possibly dependent on vehicle category. The idea of using an activity-based model to simulate time-dependent emissions is similar to the EMT. In contrast to the latter, the underlying transport in Beckx et al. (2009) does not account for congestion effects and different traffic states. Additionally, similar macroscopic emission models are typically unable to capture certain microscopic behavior accurately (see, e.g., Ahn and Rakha, 2008).

Hirschmann et al. (2010) link the microscopic traffic flow simulator VISSIM with the instantaneous emission model PHEM.<sup>1</sup> At first glance, this approach seems very promising, as it also builds the basis for the HBEFA database. In contrast to the EMT, it is not suitable for large-scale scenarios due to the computational complexity of VISSIM. In Kraschl-Hirschmann et al. (2011), the same authors attempt to develop a parametrization of fuel consumption based on average speeds of vehicles. Such parametrization could be helpful—in the future—to replace time-consuming lookups in large databases (e.g., HBEFA). However, the model would need to allow for more input variables (e.g., vehicle category, traffic state, etc.) and provide more differentiated outputs, e.g., different emission types.

In a similar study, Song et al. (2012) couple VISSIM with the emission modeling tool MOVES. They find that the VISSIM-simulated, vehicle-specific power distribution for passenger cars deviates significantly from the observed distribution, meaning that the estimated emissions also contain significant errors. Here again, the proposed model cannot be used for large-scale scenarios. Additionally, it seems questionable whether such detailed modeling will prove to be superior to less detailed models as the EMT.

Wismans et al. (2013) compare passenger car emission estimates of static and dynamic traffic assignment models. They claim that little research has been done in connecting macroscopic or meso-scopic dynamic traffic assignment models with emission models. According to the authors, static assignment models predict congestion on the wrong locations and ignore spillback effects. They argue that emission hotspots are, in consequence, also predicted at the wrong locations and/or with the wrong amplitude. To counter these disadvantages, they couple a static and a dynamic traffic assignment model with the exhaust emission model ARTEMIS. Large differences in air pollutant emissions are found and hotspot locations differ.

Hatzopoulou and Miller (2010) develop a methodology for calculating exhaust emissions, using MATSim as transport model. The approach is therefore similar to the EMT. In contrast to that study, the EMT does

---

<sup>1</sup> The PHEM model uses speed trajectories as input and was tested against the output of the EMT by Hülsmann et al. (2011).

not assume fixed exhaust emissions per time unit. It uses a more detailed calculation of emissions based on the two different traffic states: “free flow” and “stop&go”. It is, thus, able to capture congestion effects that emerge, as well as the time spent in traffic jam. Furthermore, the EMT calculates exhaust emissions for passenger cars *and* for trucks. Finally, since the methodology is based on HBEFA, it can be transferred to any scenario in Europe.

## 15.4 Emission Calculation

Air pollution is caused by different contributions of road traffic: Warm emissions are emitted while driving and are independent of the engine’s temperature. Cold-start emissions also occur during the warm-up phase and depend on the engine’s temperature when the vehicle is started. Warm emissions differ with respect to: *driving speed*, *acceleration/deceleration*, *stop duration*, *road gradient*, and *vehicle characteristics* consisting of vehicle type, fuel type, cubic capacity, and European Emission Standard Class (André and Rapone, 2009). Cold emissions differ with respect to: *driving speed*, *distance traveled*, *parking time*, *ambient temperature*, and *vehicle characteristics* (Weilenmann et al., 2009).

Currently, the emissions contribution to MATSim considers all differentiations above marked in *italic*. Road gradient and ambient temperature are not considered; gradient is always assumed to be 0%, and ambient temperatures are assumed to be HBEFA average. In addition to warm and cold-start emissions, evaporation and air conditioning emissions also result from road traffic. At the moment, these are not considered in the emission modeling tool, because they contribute little to the overall emission level.

The calculation of warm emissions is composed of two steps:

1. deriving *kinematic characteristics* from the simulation, and
2. combining this information with vehicle characteristics to extract emission factors from the HBEFA database.

In the first step, driving speed, as well as stop duration (and possibly an approximation of acceleration/deceleration patterns), is captured by a mapping of MATSim’s dynamic traffic flows to HBEFA traffic states. These traffic states, namely “free flow”, “heavy”, “saturated”, and “stop&go”, have been derived from typical driving cycles, i.e., time-velocity profiles. A parametrization of these profiles led to the definition of these traffic states, which depend on speed limit, average speed, and road type. Thus, typical emission factors for a specific traffic state on a specific road segment can be looked up in the HBEFA database. In MATSim, neither the location on a road segment, nor the exact driving behavior of an agent is known (see Section 1.3). It is quite straightforward to extract agents’ travel times on the road segment which, thanks to the queuing model, also includes interactions with other agents and spillback effects. The average speed of an agent on a certain road segment is thus used to identify corresponding HBEFA traffic states, and to assign emission factors to the vehicle. As of now, the emission modeling tool considers only two traffic states: free flow and stop&go.<sup>2</sup> Each road segment is divided into two parts representing these two traffic states. The distance  $l_s$  that a car is driving in stop&go traffic state is determined by the following equation:

$$l_s = \frac{l v_s (v_f - v)}{v (v_f - v_s)}, \quad (15.1)$$

where  $l$  is the link length in kilometers from the network,  $v_s$  is the stop&go speed in  $km/h$  for the HBEFA road type,  $v_f$  is the free flow speed in  $km/h$  from the network, and  $v = \frac{l}{t}$  is the average speed on the link for the vehicle,  $t$  being the link travel time of the vehicle in the simulation. For the derivation of Equation (15.1), please refer to Kickhöfer (2014). The distance that the car is driving in free flow traffic state is then simply the remaining link length  $l_f = l - l_s$ . The interpretation of this approach: Cars

<sup>2</sup> Simplified because the difference between traffic states—free flow, heavy, and saturated—emission factors are only marginal. In contrast, emission factors for stop&go are roughly twice as high.

drive in free flow until they have to wait in a queue. Stop&go traffic state applies only in the queue. According to the MATSim queue model presented in Section 1.3, a queue emerges if demand exceeds capacity of a road segment, which can also result in spill-back effects on upstream road segments. The length of the queue is, thus, approximated by Equation 15.1, where the average speed  $v$  on a link is the only exogenous variable.

For the second step, agent-specific vehicle attributes are needed. They are usually obtained from survey data during the initial population synthesis. The vehicle attributes typically comprise: vehicle type, age, cubic capacity and fuel type. Because MATSim keeps socio-demographic information throughout the simulation process, it can be used at any time for reference in the detailed HBEFA database. Additionally, the emission modeling tool is designed in such way that fleet averages are used, whenever no detailed vehicle information is available.

The calculation of cold-start emissions is, again, composed of two steps:

1. deriving *parking duration* and *accumulated distance* from the simulation, and
2. combining this information with vehicle characteristics in order to extract emission factors from the HBEFA database.<sup>3</sup>

Parking duration refers to the time a vehicle is not moved *before* cold-start emissions are produced. It is calculated by subtracting an activity's start time from the same activity's end time and by checking if the trip to and from the activity is performed by car. Emission factors in HBEFA are differentiated by parking duration in one hour time steps from 1 hour to 12 hours. After 12 hours, the vehicle is assumed to have fully cooled down. The accumulated distance refers to the distance a vehicle travels *after* a cold start. According to HBEFA, there are different cold-start emissions for short trips less than 1 kilometer and for longer trips equal to or greater than 1 kilometer. In reality, cold-start emissions are emitted along the route after a cold start; at this time, the emission modeling tool maps the short trip emissions to the road segment where the engine is started, and, if applicable, additional emissions to the road segment where the accumulated distance exceeds the first kilometer. Overall, cold-start emission factors increase with parking duration and accumulated distance; they also depend on vehicle attributes. The lookup for this information is identical to the one described for warm emissions.

In order to further process warm and cold-start emissions, so-called *emission events* are generated during the simulation in a separate events stream. The definition of emission events follows the MATSim framework that uses events for storing disaggregated information in XML format. The following section provides more information on the EMT's software structure.

## 15.5 Software Structure

The information in this section refers to code that can be found in the MATSim repository. In the following, the software structure of the EMT at revision 30058 is described. For information on how to use the tool, please use the entry points listed at the beginning of this Chapter 15.

Figure 15.1 shows the simplified software structure of the EMT. The core of the tool is the `EmissionModule` which needs to be created before the simulation starts. There are also two public methods that must be called: `createLookupTables()` and `createEmissionHandler()`.

The former creates lookup tables from input data that has to be exported from the HBEFA database. The path to these input files can be configured in the `EmissionsConfigGroup`. Mandatory input are files for the creation of `roadTypeMapping`, `emissionVehicles`, `avgHbefaWarmTable`, and `avgHbefaColdTable`. The first lookup table maps road types from the MATSim network to HBEFA road types. For this mapping, it is necessary to classify the network road types into HBEFA categories; this requires some transport engineering knowledge. The second lookup table defines the vehicle attributes of every

---

<sup>3</sup> Please note that HBEFA provides cold-start emission factors only for passenger cars. Freight traffic therefore only produces cold-start emissions of passenger cars.

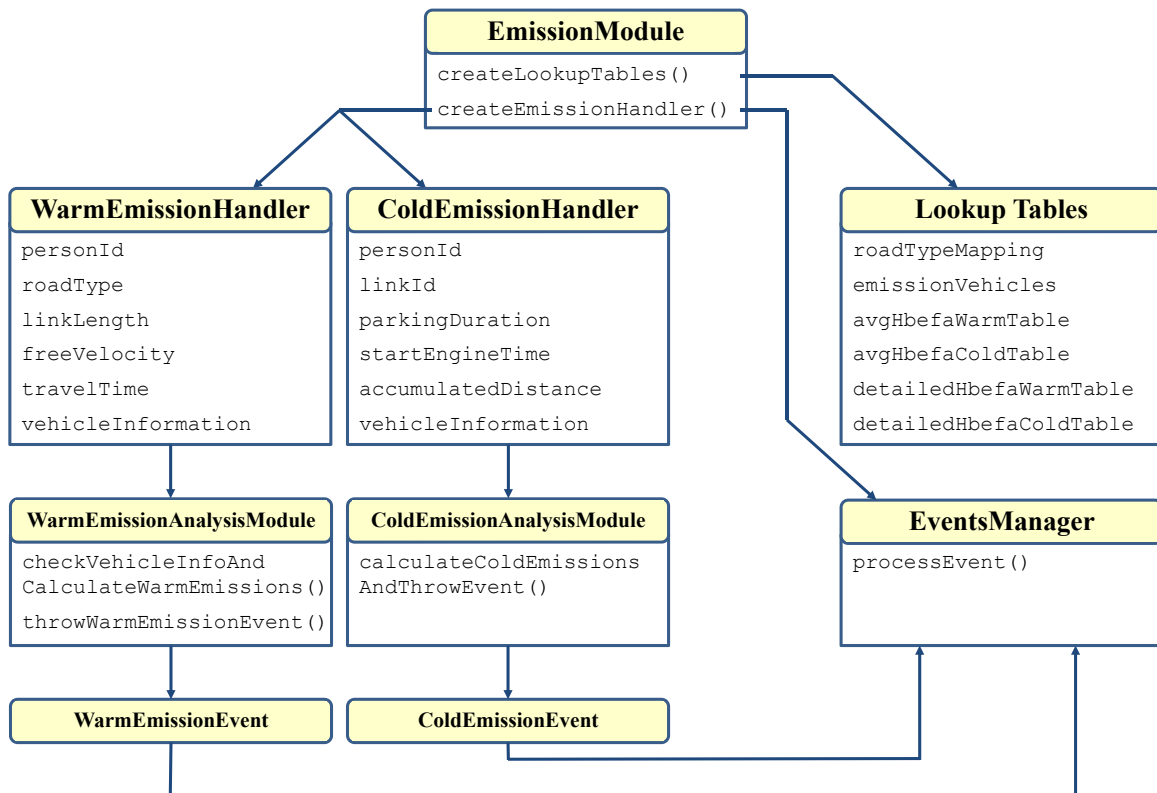


Figure 15.1: Software structure of the emission modeling tool.

owner in the population. It should therefore be generated during the population synthesis process. If no detailed information is available, the vehicle lookup table still needs to specify whether the vehicle is a car or a truck. The current implementation uses the MATSim vehicle interface `Vehicles` as container for storing the relevant data in `VehicleType`.<sup>4</sup> The last two mandatory lookup tables (`avgHbfaWarmTable` and `avgHbfaColdTable`) provide warm and cold emission factors in  $g/km$ , respectively. The data is stored using a unique key. For the construction of this key, information from `roadTypeMapping` and `emissionVehicles` is needed, as well as information derived from the simulation as described in Section 15.4. The latter information is depicted in Figure 15.1 as variables of the two classes `WarmEmissionHandler` and `ColdEmissionHandler`. These two handlers implement several MATSim `EventHandler` interfaces to extract necessary information from the simulation. After gathering this information, the `WarmEmissionHandler` asks its `WarmEmissionAnalysisModule` to reconstruct the key and look up the emission factors in the respective table. Similarly, the `ColdEmissionHandler` asks the `ColdEmissionAnalysisModule`. These analysis modules then create `Warm/ColdEmissionEvents`, which follow the MATSim `Event` interface definition. Finally, the resulting events stream is written in a joint emission events file by a separate `EventsManager`.

For the calculation of emissions dependent on agent-specific vehicle characteristics, `emissionVehicles` must contain that specific information, the corresponding flag in the `EmissionsConfigGroup` needs to be switched on, and detailed emission factor tables also need to be exported from HBEFA and provided to the `EmissionModule` with two additional input files: `detailedHbfaWarmTable` and `detailedHbfaColdTable`.

<sup>4</sup> Please note that vehicle information provided to the `EmissionModule` is *only* used for storing data on individual vehicle characteristics and other information will be omitted by the simulation.





# 16

## Noise

**Author** (of documentation): Kai Nagel

### 16.1 Basic Information

**Entry point to documentation:**

<https://github.com/matsim-org/matsim-libs/tree/master/contribs> → noise

**Invoking the module:**

<https://github.com/matsim-org/matsim-libs/tree/master/contribs> → noise →  
NoiseOfflineCalculationExample class

**Selected publications:**

Kaddoura et al. (2017)



# Accidents

**Author** (of this documentation): Kai Nagel

## 17.1 Basic Information

**Entry point to documentation:**

<https://github.com/matsim-org/matsim-libs/tree/master/contribs> → accidents

**Invoking the module:**

<https://github.com/matsim-org/matsim-libs/tree/master/contribs> → accidents → RunAccidents class

**Selected publications:**

Mayobre (2018)



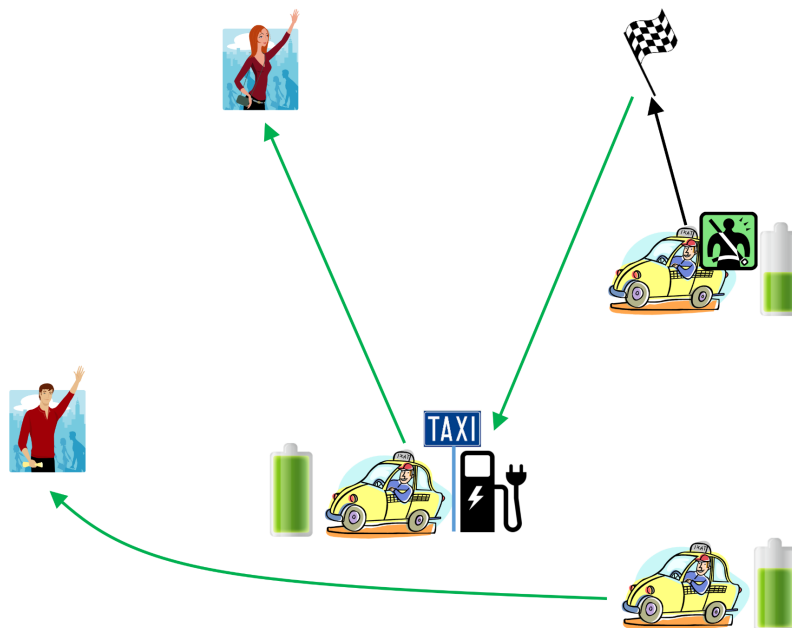
**Part VI**

# **Extending MATSim**



## Dynamic Transport Services

Author: Michal Maciejewski



### Entry point to documentation:

<https://github.com/matsim-org/matsim-libs/tree/master/contribs> → dvrp

### Invoking the module:

No predefined invocation. Starting point(s) under <https://github.com/matsim-org/matsim-libs/tree/master/contribs> → dvrp → RunOneTaxiExample class.

### Selected publications:

Maciejewski and Nagel (2013a,b, 2014); Maciejewski (2014)

## 18.1 Introduction

The recent technological advancements in Information and Communications Technology (ICT) 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.

## 18.2 DVRP Contribution

To address the problem above, MATSim's Dynamic Vehicle Routing Problem (DVRP) 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 Vehicle Routing Problems (VRPs); one can easily extend the model even further to cover other specific cases (see Section 18.3). 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.



## 18.3 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 18.3.2), which are calculated on demand, all the VRP-related data are accessible via `VrpData`.<sup>1</sup> In the most basic setup, there are only two types of entities, namely `Vehicle`s and `Request`s. 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.

### 18.3.1 Schedule

Each `Vehicle` has a `Schedule`, a sequence of different `Task`s, such as driving from one location to another (`DriveTask`), or staying at a given location (e.g., serving a customer or waiting; `StayTask`).<sup>2</sup> A `Schedule` is where supply and demand are coupled. All schedules are calculated by an online optimization algorithm (see Section 18.6) 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`.<sup>3</sup> 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.

---

<sup>1</sup> Package `org.matsim.contrib.dvrp.data`.

<sup>2</sup> Package `org.matsim.contrib.dvrp.schedule`.

<sup>3</sup> 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).

### 18.3.2 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).<sup>4</sup> 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`).

## 18.4 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).

### 18.4.1 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:

---

<sup>4</sup> 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.

### 18.4.2 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 `EventManager`, `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 `EventManager`.

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

### 18.4.3 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 18.4.2, setting up and running this example scenario is straightforward. `RandomDynAgentLauncher` reads a network, initializes a `QSim`, then adds a `RandomDynAgentSource` to the `QSim`, and finally, launches visualization and starts simulation. The `RandomDynAgentSource` is responsible for instantiating 100 `DynAgents` that are randomly distributed over the network. The simulation ends when the last active agent becomes inactive.

## 18.5 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 18.6). 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.

### 18.5.1 Drivers

A driver is modeled as a `DynAgent`, whose behavior is controlled by a `VrpAgentLogic` that makes the agent follow the dynamically changing `Schedule`.<sup>5</sup> 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.

### 18.5.2 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.

---

<sup>5</sup> 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`<sup>6</sup>. 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 may 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.

## 18.6 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 local 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<sup>7</sup>, 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.<sup>8</sup> 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).

---

<sup>6</sup> Package `org.matsim.contrib.dvrp.passenger`.

<sup>7</sup> Package `org.matsim.contrib.dvrp.optimizer`.

<sup>8</sup> This can be achieved by using an optimizer implementing the interface `org.matsim.core.mobsim.framework.listeners.MobsimBeforeSimStepListener`.

## 18.7 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.<sup>9</sup>
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.

## 18.8 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.

<sup>9</sup> Package `org.matsim.contrib.dvrp.vrpagent`,

## 18.9 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, Yarrowonga and Mulwala, described in Chapter ?? (Ronald et al., 2015, 2014). In a recently launched project, the DVRP contribution will be used for simulation of Demand Responsive Transport (DRT) 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 18.3.2). 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.





## The “Minibus” Contribution

**Authors:** Andreas Neumann, Johan W. Joubert

### 19.1 Basic Information

**Entry point to documentation:**

<https://github.com/matsim-org/matsim-libs/tree/master/contribs> → minibus

**Invoking the module:**

<https://github.com/matsim-org/matsim-libs/tree/master/contribs> → minibus → RunMinibus class

**Selected publications:**

Neumann (2014)

### 19.2 Paratransit

Paratransit is an informal, market-oriented, self-organizing public transport system. Despite the significance of this transport mode, it is mainly unsubsidized, relying on collected fares. Paratransit systems can be categorized by route pattern and function, by driver organization, type of stops and fare type. Most case studies covered by the Neumann (2014) thesis indicate that paratransit services are mainly organized as route associations operating 8-15 seater vans on fixed routes. Most of the services run in direct competition to a public transport system belonging to a public transit authority. Such a service—minibuses with fixed routes, but without fixed schedule—is often called a jitney service. The minibuses module of MATSim is based on the most common characteristics, with the understanding that the jitney/minibus service is only one of many possible paratransit services.

The minibuses model is integrated in the multimodal multi-agent simulation of MATSim. In the model, competing minibuses operators begin to explore the public transport market, offering their services. With more successful operators expanding and less successful operators going bankrupt, a sustainable network of minibuses services evolves. In Neumann (2014), the model is verified through multiple illustrative scenarios, analyzing the model’s sensitivity to different demand patterns, transfers, and interactions of minibuses and a formal operator’s fixed train line.

The minibuses model can be applied to two different transport planning fields. First: in the simulation of real paratransit targeting the inner workings of different paratransit stakeholders’ relationships, the model can create “close-to-reality” minibuses networks in a South African context. Neumann et al. (2015) gives an in-depth presentation of the module application and South African paratransit in general. Given

the informal and emergent nature of minibus paratransit in developing countries, routes, schedules and fares are usually not published; they can only be captured in the tacit knowledge of operators and frequent users. Applying the minibus model has proven valuable in gaining a better understanding of how routes evolve. Instead of imposing routes and schedules *on* the MATSim model, as is usually the case for formal transit, the modeler observes and gets the paratransit routes as an output *from* the model. As each operator aims to maximize their profit, the resulting network often favors the operators’ business objectives, instead of the connectivity and mobility of the mode’s users. This model feature accurately captures route-forming behavior in the South African case, where commuters are often required to take multiple, longer trips instead of direct trips.

Second, the same model provides a demand-driven approach to solving a formal transit authority’s network design problem; it can be used as a planning tool for the optimization of single transit lines or networks. For more details on the second form of application, see Section 19.3.

For further reading: Neumann (2014) provides an understanding of the underlying principles of paratransit services, namely minibus services, its stakeholders, fares, route functions, and patterns. Furthermore, it contains an in-depth description of the minibus model, its theoretical background, and its application to illustrative scenarios, as well as real world examples. The website of MATSim also hosts latest implementation documentation at <http://matsim.org/doxygen>.

## 19.3 Network Planning or Solving the Transit Network Design Problem with MATSim

A public transport system’s success depends primarily on its network design. When transport companies try to optimize a line using running costs as the main criteria, they quickly find that demand must be taken into consideration. The best cost structure is unsustainable if potential customers leave the system and opt for alternatives, like private cars. The basic problem to solve: find sustainable transit lines offering the best possible service for the customer.

More specifically,

- the customer’s demand side asks for direct, uncomplicated connections, and
- the operator’s supply side asks for profitable lines to operate.

Informal public transit systems around the world, often referred to as paratransit, are examples of market-oriented, self-organizing public transport systems. For an in-depth coverage of paratransit, see Section 19.2, with references. Despite the significant and increasing importance of this transport mode, it is mainly unsubsidized and relies only on collected fares. Thus, the knowledge of paratransit—and its ability to identify and fill market niches with self-supporting transit services—provides an interesting approach to solving a formal public transit company’s network design problem.

The minibus module of MATSim provides a demand-driven approach to solving a formal transit authority’s network design problem; it can be used as a planning tool for the optimization of single transit lines or networks. In the Neumann (2014) thesis, the model was applied to two different planning problems of the Berlin public transit authority Berliner Verkehrsbetriebe (BVG). In the first scenario, the model constructed a transit system, from scratch, for the district of Steglitz-Zehlendorf. The second scenario analyzed the Tegel airport closure impact on BVG’s bus network. Apart from Tegel itself, the rest of the bus network was unaffected by the airport closure. The resulting minibus model transit system resembled the changes BVG had scheduled for Tegel’s closure.

In conclusion, the minibus model developed in the thesis automatically adapted supply to demand. The model not only grew networks from scratch, but also tested an existing transit line’s sustainability and further optimized the line’s frequency, time of operation, length, and route. Again, the optimization process was fully integrated into the behavior-rich, multi-agent simulation of MATSim, reflecting passen-

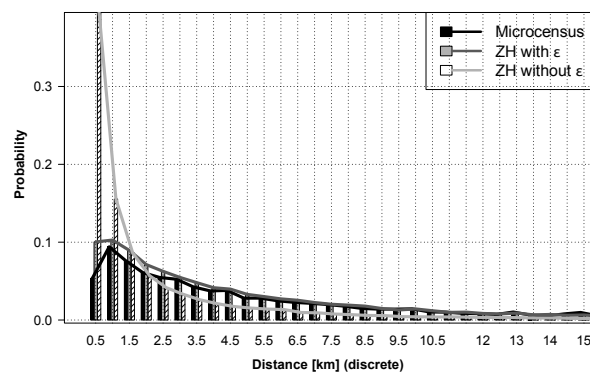
ger reactions, as well as those from competing transit services and other road users. Thus, the minibus model can be used, along with more complex scenarios, like city-wide tolls or pollution analyses.



# 20

## Destination Innovation

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



### 20.1 Basic Information

Entry point to documentation:

<https://github.com/matsim-org/matsim-libs/tree/master/contribs> → locationchoice

Invoking the module:

<https://github.com/matsim-org/matsim-libs/tree/master/contribs> → locationchoice →  
RunLocationChoiceBestResponse, RunLocationChoiceFrozenEpsilons classes

Selected publications:

Horni et al. (2012b); Horni (2013)

### 20.2 Introduction

Generally speaking, destination choice represents an optimization problem, where every agent searches for his or her optimal destination according to an objective function, subject to various constraints such as the agent's travel time budget—as well as interactions with other agents—while competing for space-time slots in the infrastructure. The MATSim destination innovation module provides a problem-tailored heuristic algorithm to solve this problem.

MATSim's iterative base requires a mechanism (the main component of the destination innovation module), ensuring consistent probabilistic choices over the course of iterations.

Unobserved heterogeneity, usually dominant in destination choice, is captured in the adaptable objective function by random error terms (Horni et al., 2012b; Horni, 2013).

As well as considering competition for road infrastructure, the destination choice module can also be configured for activities infrastructure (for example, at shopping malls' parking lots) as shown in Section 20.3.5 and by Horni et al. (2009).

## 20.3 Key Issues in Developing the Module

Key issues of integrating destination innovation into MATSim include behavioral and algorithmic problems. On the behavioral side, specification of choice sets for model estimation has not yet been solved. On the algorithmic side, as mentioned above, destination innovation is, in principle, an ordinary optimization problem. However, as agents interact, and choices are embedded in a highly dynamic context, the problem becomes complex, particularly because targeted scenarios are usually large-scale. Thus, as in real-world optimization problems, solutions must be based on problem-tailored heuristics (Michalewicz and Fogel, 2004). Construction of a search space and subsequent evaluation of the search space's elements are important MATSim destination innovation components.

The main component however, is a mechanism to generate consistent random draws over iterations necessary to include the objective function's error terms (see next Section 20.3.1). This mechanism is also applicable to other choice dimensions.

### 20.3.1 Error Terms

As described in Chapter 22.6, MATSim—as a utility-maximizing model—is related to the discrete choice framework, meaning that this framework can productively guide the MATSim utility function specification. Utility in discrete choice models is composed of a deterministic part and a random error term representing the unobserved heterogeneity, i.e., it subsumes, both truly, i.e., inherently random, decisions and the modeler's missing knowledge about the choice and its context.

In MATSim, the utility function for route, mode and time innovation does not contain an explicit random error term (yet). This is at least partially compensated through replanning stochasticity, in Chapter 22.6 denoted by the scale parameter  $\mu$  and  $\eta$ . An example for this might be: route and time choices are usually subject to significant competition. The co-evolutionary algorithm of MATSim, detailed below, essentially assigns the resources in a random manner to the persons. For example, two identical persons may end up with different routes, according to the order in which they undergo the replanning. Essentially, this means that an (implicit) random term is present in the choice making.

The above, however, does not add enough unobserved heterogeneity to destination choice. Further problems might, or might not, appear when trying to interpret this randomness, since it is added implicitly and somewhat unsystematically. Thus, an explicit random error term  $\varepsilon_{n\ell q}$  for every person  $n$ , alternative  $\ell$  and activity  $q$ , held stable over the iterations, is added to the scoring function during the running of the destination innovation module (Horni, 2013). Research about the necessity of error terms for the remaining choice dimensions is required, as discussed in Section ??.

### 20.3.2 Quenched Randomness

Due to random error terms, discrete choices are quantified by probabilities; for example, for the logit model, as  $p_{n\ell q} = \exp(V_{n\ell q}) / \sum_{j \in L} \exp(V_{njq})$ , where  $V_{n\ell q}$  is person  $n$ 's systematic utility of alternative  $\ell$  for activity  $q$ . When drawing from the distribution specified by  $p_{n\ell q}$  for a population, the aggregate choices are reproduced. This is basically also true when applied in iterative frameworks.

However, iterative frameworks are usually associated with some kind of learning or relaxation mechanism, which is heavily distorted by repeatedly and randomly drawing from  $p_{n\ell q}$  in every iteration. In this case, the  $\varepsilon_{n\ell q}$  effectively fluctuate from iteration to iteration, which is disastrous for the algorithm's convergence and behaviorally implausible.

Instead, random error terms  $\varepsilon$  must remain fixed from iteration to iteration. The optimization is then performed as a deterministic search, based on the resulting utilities  $U_{n\ell q}$ , i.e., an alternative  $\ell$  for person  $n$ ; activity  $q$  is selected as

$$\operatorname{argmax}_{\ell \in \text{choice set}} U_{n\ell q} = V_{n\ell q} + \varepsilon_{n\ell q}.$$

This includes, via the systematic part  $V_{n\ell q}$ , the disutility of traveling to destination  $\ell$  for activity  $q$ .

As stated above, random error terms must remain the same over the iterations (also discussed in Chapter 22.6). In physics, this approach would be called “quenched” (sometimes also “frozen”) randomness; all randomness is computed initially and then attached to particles or destinations, rather than instantaneously generating it, which would be called “annealed” randomness. Two natural approaches for implementing quenched randomness are as follows:

- (a) Freezing the applied *global* sequence of random numbers, meaning that a Monte Carlo method with the same random seed is used before and after introduction of a policy measure and over the course of iterations. Thus, error terms should come out the same way *before* and *after* the introduction of the policy measure. Differences in the outcome can thus be directly attributed to the policy measure.
- (b) Computing and storing a separate  $\varepsilon_{n\ell q}$  for every combination of person  $n$ , alternative  $\ell$  and activity  $q$ .

Both strategies have flaws. Approach (a) is only an option if one is certain about every single aspect of the computational code. Literally, one additional random number, drawn in one run, but not in the other, completely destroys the “quench” for all decisions computed later in the program. Consistency is thus hard to achieve, especially in parallel or even distributed computing environments; substantial machinery is necessary to ensure consistent choices. In a modular environment, as in MATSim, designed for external plugging-in of users' own modules—possibly drawing their own random numbers—the danger of destroying the quench is prohibitively high and thus approach (a) is impractical.

Approach (b) is certainly more robust. However, for large numbers of decision makers and/or alternatives, storing error terms is difficult. For destination innovation, one quickly has  $10^6$  decision makers and  $10^6$  alternatives, resulting in  $4 \cdot 10^{12}$  Byte = 4TB of storage space.

One may argue that this should not be a problem, since a normal person will rarely consider more than the order of a hundred alternatives in their choice set, reducing the computational problem. Aside from the necessity of storing every decision maker's choice set, this converts the computational problem into a conceptual one, since a good method to generate choice sets then needs to be found. With more conceptual progress, this may eventually be an option; at this point, a conceptually simpler approach is preferred.

The solution developed below is generally applicable in econometric microsimulators. The same *stable* error term can be *re-calculated* on the fly by using stable random seeds  $s_{n\ell q} = g(k_n, k_\ell, k_q)$ , containing uniformly distributed random numbers associated with  $k$ ,  $\ell$ , and  $q$ . That is, for each person  $n$ , a random number  $k_n$  is generated and stored; the same is done with each destination  $\ell$ . Value for the activity  $q$  can be derived from its index in the plan, possibly combined with the person's value  $k_n$ . This reduces the storage space dramatically, from  $N_q \cdot N_n \cdot N_\ell$  to  $N_q(N_n + N_\ell)$ , where  $N_n$  is the number of persons or agents and  $N_\ell$  is the number of destinations and  $N_q$  is the average number of discretionary activities in an agent's plan. This means that storage space is reduced to approximately  $2 \cdot 4 \cdot 10^6$  Byte = 8MB, which can be easily stored on any modern machine.

Distribution of these seeds is essentially irrelevant; any error term distribution can be generated from any basic seed distribution. In the current version,  $g(k_n, k_\ell, k_q) = (k_n + k_\ell + k_q) \times v_{max}$  is used.  $v_{max}$  is the maximum (long) number that can be handled by the specific machine.

To evaluate utility for a person  $n$  visiting the destination  $\ell$  for activity  $q$ , a sequence of Gumbel-distributed random numbers  $seq_{n\ell q}$  is generated on the fly for every person-alternative-activity combination using the seed  $s_{n\ell q}$ . Some random number generators have problems in the initial phase of drawing, e.g., the first couple of random numbers are correlated or never cover the complete probability space. As in our procedure, the random number generator is constantly re-initialized; for these technical reasons, the error term  $\varepsilon_{n\ell q}$  is not derived from the first element, but from the  $m^{th}$  element of the sequence  $seq_{n\ell q}[m]$ . Here,  $m$  is set to 10. This procedure is valid, as the set of all  $m^{th}$  elements of all different sequences is also a pseudo-random sequence, following the same distribution as the sequences  $seq_{n\ell q}$ ; clearly, *true* random number generators relying on physical phenomena, such as hardware temperature, are not applicable.

### 20.3.3 Search Space Construction and Evaluation

MATSim destination innovation is based on best-response, rather than random mutation; in every iteration, the best current alternative, *including* the  $\varepsilon_{n\ell q}$ , is chosen. This works as long as inter-iteration changes are small, which usually happens, given by the relatively small share of agents who re-plan. The best-response approach is adopted due to the usually huge number of alternatives in combination with the search space characteristics. The discrete search landscape is characterized by random noise, because error terms are not spatially correlated (see Figure 20.0(a)). For such problems—as opposed to continuous landscapes (see Figure 20.0(b))—efficient search methods, such as local search methods, generally do not work.

When searching for the best choice, the large number of alternatives—prohibiting exhaustive search—is restrained as follows (for the detailed derivation see Horni, 2013, p.51 ff.). It is assumed that travel costs are always negative and that a person drops activities with negative net utility. Then, the maximum potential travel effort a person is willing to invest is constrained by the maximum error term per person and activity. This approach is promising, as very large values for Gumbel-distributed variables are rare, meaning that a huge space must be searched for only a few persons.

This search space reduction saves a great deal of computation time; however, it is still unfeasible and further speed-ups are necessary. Most computation time is due to travel time calculation, i.e., due to routing, for evaluation of the alternatives in the search space. To reduce these huge routing costs, the Dijkstra (Dijkstra, 1959) routing algorithm is not only applied forward—providing one-to-all travel times—but also *backwards*, using an average estimated arrival time as initial time. This is an approximation; thus, a *probabilistic* best response is applied, justified by the assumption that, during the course of the iterations, the probabilistic choice will reduce the errors incurred by approximating travel times.

With this procedure, the required computational effort is dramatically reduced, allowing application of destination innovation to large-scale scenarios.

### 20.3.4 Destination Choice Set Specification

Choice set specification is natural for choices with few alternatives; but in contrast, for problems with a large universal choice set, specifying individual choice sets becomes a challenging computational and behavioral issue. This is particularly true for spatial choices like destination or route choice (e.g., Pagliara and Timmermans, 2009; Thill, 1992; Schüssler, 2010; Frejinger et al., 2009). Estimates are sensitive to choice sets; at the same time, no established choice set definition procedure exists for spatial problems. This means that choice sets and, hence, estimates are dependent on the modeler.

An important extension of the standard discrete choice modeling approach to treat this problem is formed by stochastic choice set models, founded by Manski (1977); Burnett and Hanson (1979); Burnett



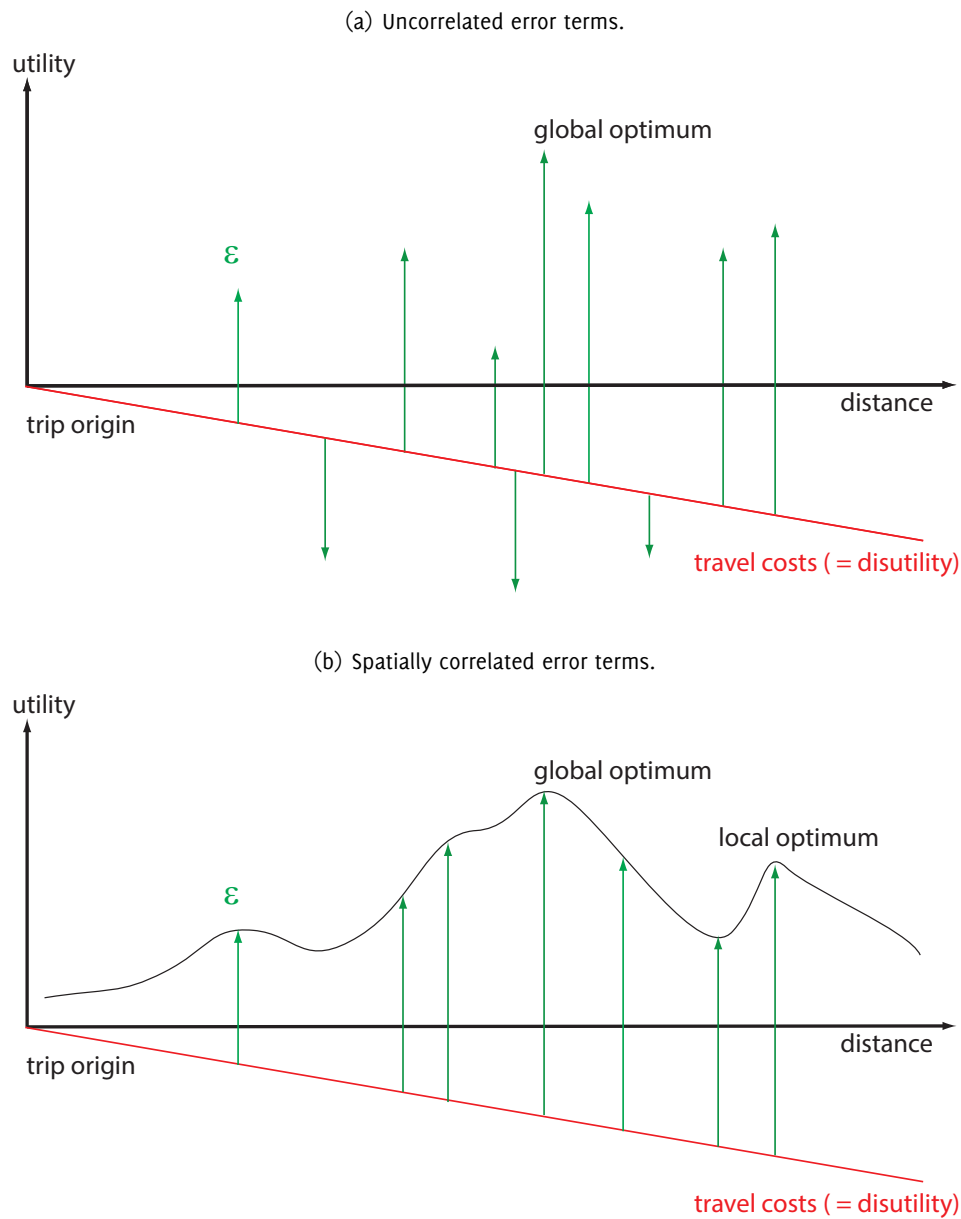


Figure 20.1: Search space: The search algorithm must be able to handle correlated, as well as uncorrelated, error terms as given by the MNL model. Local search methods, such as hill-climbing algorithms are only able to handle continuous search spaces; thus, for situation (a), a best-response global search algorithm is required.

(1980); these integrate the choice set formation step into the estimation procedure by jointly estimating choice set selection and selection of a particular alternative of this choice set (Manski, 1977; Ben-Akiva and Boccara, 1995). Probabilistic choice set formation is conceptually appealing; choice sets are, in principle, not restrained a priori by exogenous criteria, as in standard choice set specification. However, the procedure is generally associated with combinatorial complexity, making it computationally intractable. As a consequence, practical approaches also require mechanisms to reduce complexity of the choice set specification problem (e.g., Ben-Akiva and Boccara, 1995, p.11). Zheng and Guo (2008), for example, make the moderate assumption of continuous store choice sets (i.e., sets without “holes”) around the trip origin, while Ben-Akiva and Boccara (1995)’s random-constraints model exploits additional information on alternatives’ availability for individuals.

In conclusion, the destination innovation set specification problem is still unsolved, meaning that estimated models can only be fully consistently applied for the region where the model was estimated. For MATSim, destination choice model estimation efforts are reported in Horni (2013, Chapter 5).

### 20.3.5 Facility Load

The influence of interaction in *transport* infrastructure for people’s route and departure time choice was recognized almost a century ago (e.g., Pigou, 1920; Knight, 1924; Wardrop, 1952). It can also be reasonably assumed that agent interaction in *activities* infrastructure affects travel choices (Axhausen, 2006). Marketing science provides ample evidence that agent interactions influence utility (positively or negatively) of performing an activity (Baker et al., 1994, p.331), (Eroglu and Harrell, 1986; Eroglu and Machleit, 1990; Eroglu et al., 2005; Harrell et al., 1980; Hui and Bateson, 1991; Pons et al., 2006).

In Horni et al. (2009), based on the Zürich scenario, a model is presented introducing competition for activity infrastructure space-time slots. The actual load is coupled with time-dependent capacity restraints.

Activity location load, computed for 15 minute time bins, is derived from events delivered by the mobsim. The load of one particular iteration, combined with time-dependent activity location capacity restraints, is considered in the agents’ choice process of the succeeding iteration. In detail, this means that the utility function term  $S_{dur,q}$ , described above, is multiplied by  $max(0; 1 - f_{load\ penalty})$ , penalizing agents dependent on the load of the location they frequented.  $f_{load\ penalty}$  is a power function; this has proved to be a good choice for modeling capacity restraints (remember that the well-known cost-flow function by U.S. Bureau of Public Roads (1964) is a power function). To introduce additional activity location heterogeneity, an attractiveness factor  $f_{attractiveness}$  is introduced, defined to be logarithmically dependent on the store size given by the official workplaces census.

Also for demonstration purposes, capacity restraints are exclusively applied to shopping locations; in principle, leisure activity locations could be handled similarly. However, deriving capacity restraints for leisure activity locations is expected to be much more difficult than for shopping locations, because far less data is available for leisure locations and capacity restraints vary much more between different leisure locations than between different shopping activities (hiking versus going to the movies might be a good example).

The model allows assignment of individual time-dependent capacities to the activity locations. For the sake of demonstration, the capacities of all shopping facilities can be set equal, where their values can be derived from the shopping trip information given in the Swiss microcensus (Swiss Federal Statistical Office (BFS), 2006). The total daily capacity is set so that the activity locations located in the Zürich region satisfy the total daily demand with a reserve of 50%. In detail, the capacity restraint function for a location  $l$  is as follows:

$$f_{load\ penalty,\ell} = \alpha_l \cdot \left( \frac{load_\ell}{capacity_\ell} \right)^{\beta_\ell}$$

with  $\alpha_\ell = 1/1.5^{\beta_\ell}$ ,  $\beta_\ell = 5$ .  $f_{load\ penalty,\ell}$  is the penalty factor for location  $\ell$  as described above.

Simultaneous computation of all agents' score reduction avoids the last-record problem discussed in Vovsha et al. (2002). There, a sequential choice process is proposed; alternatives are removed from later travelers' choice sets if locations are already occupied by earlier travelers. Thus, travelers' order is specified arbitrarily; the last-record problem (last travelers must go a long distance to find an available location) is significant when modeling heterogeneous travelers.

As expected, the constrained model improves result quality by reducing the number of implausibly overcrowded activity locations.

## 20.4 Application of the Module

The destination innovation module has been successfully applied for the Zürich scenario (Chapter ??), as reported in Horni (2013, p.99), for the Tel Aviv model (see Chapter ??) and for the MATSim 2030 project. Figure 20.2 and Figure 20.3 show that, through error term scaling, distance distributions can be nicely fitted, decreasing count data error.

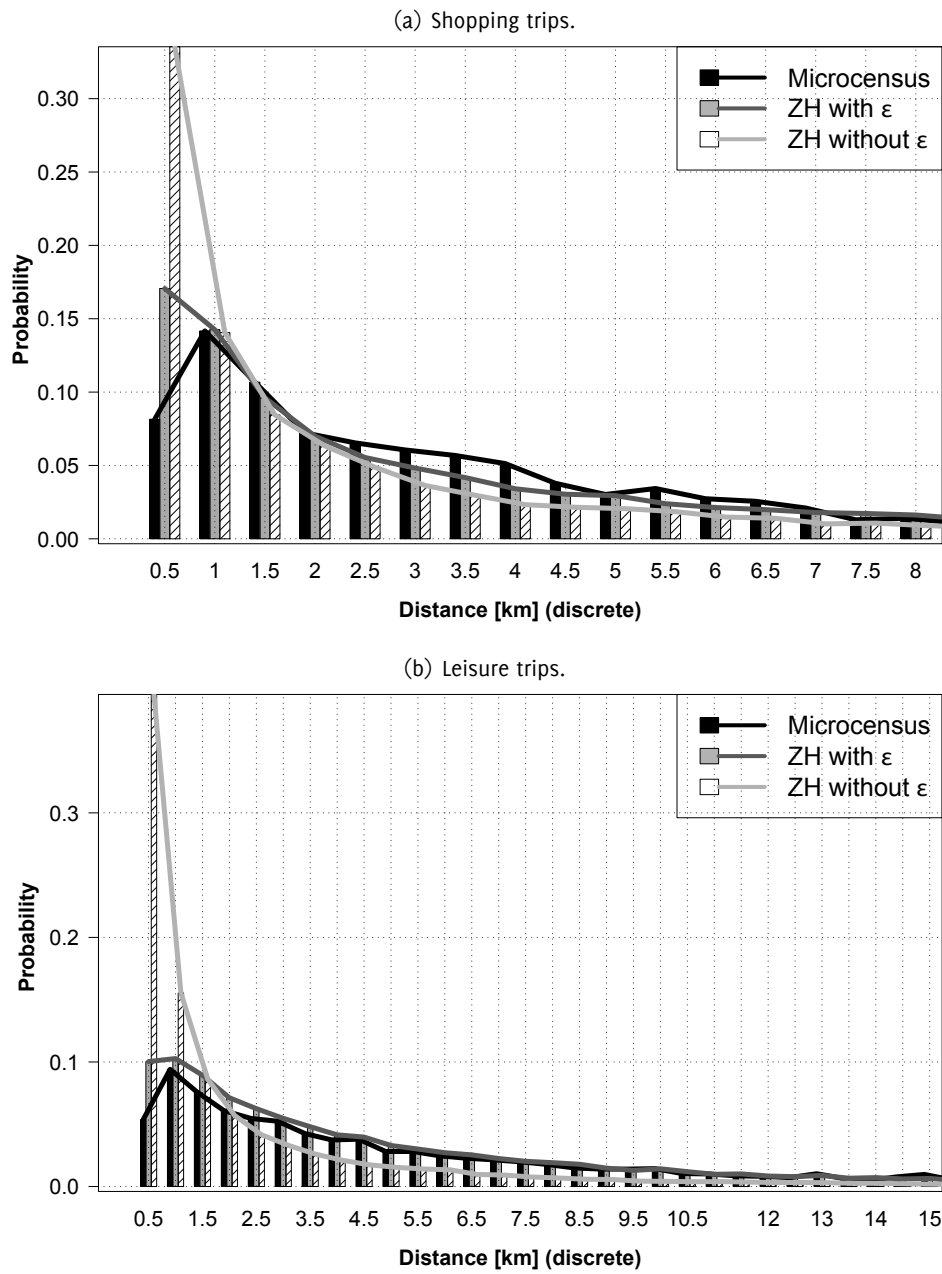


Figure 20.2: Error term runs for the Zürich scenario.

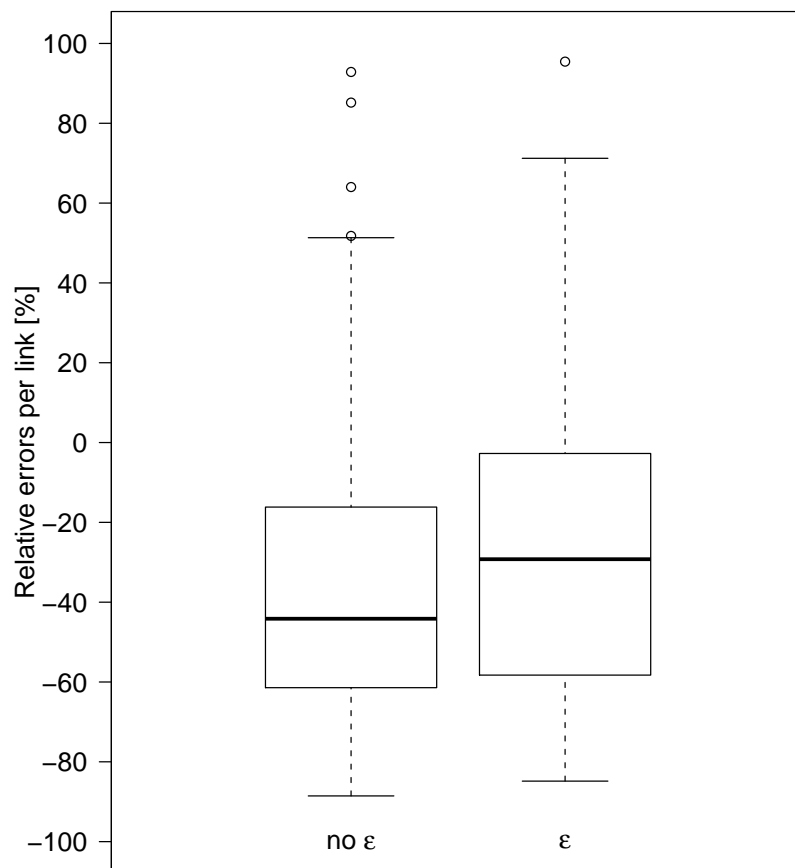


Figure 20.3: Daily traffic volumes for 123 links compared to traffic counts. Per link  $k$  the relative error is used, i.e.,  $(vol_{simulated,k} - vol_{counted,k})/vol_{counted,k}$ .

## 20.5 The Module in the MATSim Context

The destination innovation module explicitly incorporates unobserved heterogeneity through random error terms; the standard MATSim utility function, however, does not contain error terms. Randomness measured in empirical data is included implicitly through the simulation process stochasticity, including possible randomness in the choice itself. For destination innovation, this has led to a dramatic underestimation of total travel demand, making inclusion of unobserved heterogeneity inevitable. Clearly, the problem is the impossibility of making all choices at the same level; destination choice is conditional on mode choice which, in turn, is conditional on route choice. Hierarchical choice modeling has clearly showed that randomness, expressed by the logit model scale parameter, needs to be larger in higher level decisions. This chapter addresses replacing the need for more randomness in the choice model by directly including randomness into the utility function; that randomness must be quenched, otherwise the iterative procedure will just average it out. Whether the standard utility function might also profit from the innovations made for this module should be a topic for future research .

MATSim replanning offers different strategies to adapt plans, ranging from random mutation via approximate suggestions to best response answers. Destination innovation is based on best response to handle the sheer size of the alternatives set.

Although the destination innovation utility function is based on discrete choice framework, some conceptual differences about the common discrete choice models application persist. As shown above, there is no drawing from discrete choice models, but instead, maximization of an iteration-stable utility function. The set of alternatives is not necessarily limited a priori; thus, we use the notion of a search space and not of a choice set here.

## 20.6 Lessons Learned

Two interesting lessons were learned while developing the destination innovation module: first, a lesson on preferences and space interdependence and the necessity to evaluate them in combination. When looking at distance distributions (e.g., Figure 20.2) one might think that the functional form directly represents the preferences, but this is not necessarily the case. In our simulations, it is the result of a *linear* travel disutility, but applied in geographic space, where number of opportunities increases with the *square* of the radius, in other words, with the *square* of travel distance. A similar emergent effect appears when scaling random error terms. Although both negative and positive error terms are enlarged and the average remains stable, distribution gets more skewed toward the tail; for agents' choices, maximum values—not average values—are relevant.

The second lesson concerns simulation results' variability. Although random elements are not present only in destination choice, it was the largest contributor of endogenous variability when it was developed, necessitating the experiments presented by Horni et al. (2011a) (see also Section ??).

## 20.7 Further Reading

The main information source is Horni et al. (2012b); Horni (2013); technical details and documentation are available at Horni (2016) and in javadoc. Further reading related to destination choice is: Horni et al. (2013), for parking, or Horni et al. (2012a), about coupling customers' and retailers' choices or, in other words, supply and demand.

# 21

## 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.

### 21.1 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 `ControllerListeners`.

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 MobsimListeners in a similar way as this is possible with the ControllerListeners 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.

## 21.2 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 controller object:
    Controller controller = new Controller( scenario ) ;
    controller.doZzz(...) ; // (**)
    ...

    // run the iterations:
    controller.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 Controller, by a syntax explained below.



## 21.3 MATSim Extension Points: General

This chapter (Chapter 21) describes what could be called the Service Provider Interface (SPI) 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 `Controller` class. Essentially, one would override the methods calling the `mobSim`, the `scoring`, and/or the `replanning`, as explained in Section 21.1. This is now discouraged. While this pattern worked when 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 `PublicTransportController`, an `EmissionsController`, a `RoadPricingController` and an `OTFVisController`, 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. 21.6 it is possible to combine modules while keeping their respective dependencies independent.

## 21.4 Extension Points Related to Scenario

### 21.4.1 Customizable and ObjectAttributes

`Attributable`, as a facility to attach arbitrary information to MATSim data types, was already discussed in Section 10.1. This section will discuss two preceding approaches, `ObjectAttributes` and `Customizable`. In our opinion, they are no longer needed for new developments, but it may be necessary to understand them when reading older code.

**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.

**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

```
attrs.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`.

### 21.4.2 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.

## 21.5 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 frame-

work. 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.)

## 21.6 Configuring the Controller (= inject syntax)

`Controller` 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 `Controller` class is now `final`. Please do not start from old examples that use such inheritance from `Controller`; 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 `Controller` approximately is

```
controller.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.

### 21.6.1 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 than injection of the constructor, but more concise.

### 21.6.2 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.

### 21.6.3 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.

### 21.6.4 MATSim default bindings

The MATSim default bindings can be found from ControllerDefaultsModule, see [matsim.org/javadoc](https://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.

### 21.6.5 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 preferable 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 typically 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 construction 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.

### 21.6.6 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.addControllerListenerBinding()...;
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 Persons or Links.

### 21.6.7 ControllerListener: Handling Controller Events

ControllerListeners are called at the transitions of the MATSim loop (Figure 21.1), where so-called ControllerEvents are fed to the listeners.

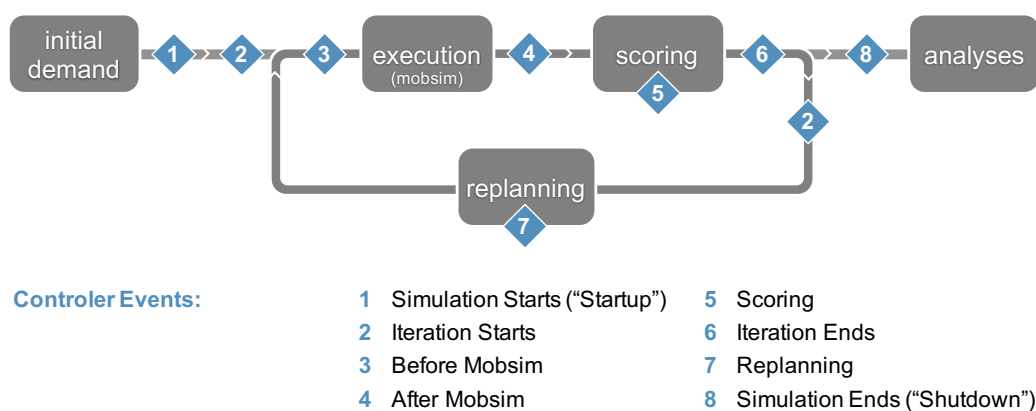


Figure 21.1: Controller events.

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

A sample listener might look as follows.

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

ControllerListeners are called in undefined order, meaning that AControllerListener may only rely on the computation of BControllerListener if BControllerListener makes that computation in an earlier transition. For instance, if BControllerListener is a StartupListener and loads data into a Map on startup, AControllerListener 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.

Please check the documentation of ControllerListener (see <http://matsim.org/javadoc> → main distribution) for more details and [RunControllerListenerExample](#) in [matsim-code-examples](#) for examples.

### 21.6.8 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 <https://github.com/matsim-org/matsim-libs/tree/master/contribs> → main distribution → `MobsimListener` for more details and [RunMobsimListenerExample in matsim-code-examples](#) for an example.

### 21.6.9 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 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 <https://github.com/matsim-org/matsim-libs/tree/master/contribs> → 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 your 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 <https://github.com/matsim-org/matsim-libs/tree/master/contribs> → main distribution → `TripRouter` for documentation and [RunTeleportationMobsimWithCustomRoutingExample in matsim-code-examples](#) for an example.

### 21.6.10 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 <https://github.com/matsim-org/matsim-libs/tree/master/contribs> → 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 Compute Unified Device Architecture, a parallel computing platform and API by NVIDIA (CUDA) 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.

### 21.6.11 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 <https://github.com/matsim-org/matsim-libs/tree/master/contribs> → main distribution → `PlanStrategy` for documentation, and [RunPluggablePlanStrategyInCodeExample](#) in `matsim-code-examples`, `RunPluggablePlanStrategyFromFileExample`, and `RunWithMultithreadedModule` for examples.

### 21.6.12 Scoring

The parameters of the default MATSim scoring function (Chapter 10) 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 <https://github.com/matsim-org/matsim-libs/tree/master/contribs> → main distribution → `ScoringFunction` for documentation, and [RunCustomScoringExample](#) in `matsim-code-examples`, `RunIndividualizedScoringExample`, `RunScenarioWithCustomScoring` and `RunOwnMoneyScoringExample` for examples.

## 21.7 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 21.7).

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(controller.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.

## 21.8 MATSim extensions

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

```
controller.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. 21.7 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.

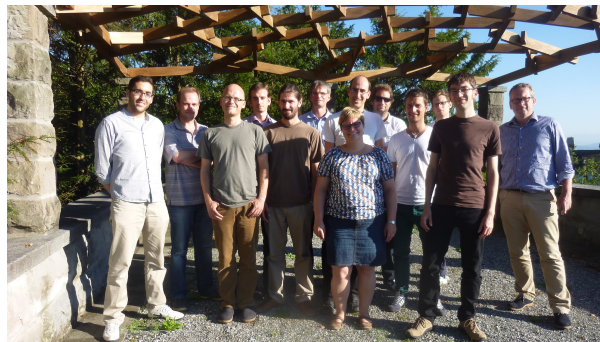
## 21.9 Conclusion

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 described 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.

## 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.

### 22.1 MATSim's Team, Core Developers Group, and Community

The **MATSim team** currently consists of three research groups and two companies:

- the VerkehrsSystemPlanung und Verkehrstelematik – The Transport Systems Planning and Transport Telematics group at TU Berlin (VSP) group at the Institut für Land- und Seeverkehr – Institute for Land and Sea Transport Systems (ILS), TU Berlin, led by Prof. Dr. Kai Nagel,
- the VerkehrsPlanung (VPL) group at the Institut für Verkehrsplanung und Transportsysteme – Institute for Transport Planning and Systems (IVT), ETH Zürich, led by Prof. Dr. Kay W. Axhausen,
- the Mobility and Transportation Planning group at the Future Cities Laboratory (FCL), 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 and research students,

- Simunto GmbH, based at Zürich, founded by a former PhD 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 22.3.2.

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 22.1).

MATSim is open-source software under the GNU General Public License version 2.0 (GPLv2). You are also very welcome to contribute to the code base as described in Section 22.6. New contributors are mentored in the beginning to become familiar with the project and the coding conventions.



Source: ©Dr. Marcel Rieser, Simunto GmbH

Figure 22.1: MATSim events and community.

## 22.2 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`.<sup>1</sup> 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 21, 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 22.3.2), usually after having discussed them in the issue tracker (<http://matsim.org/issuetracker>), in the MATSim committee, or at a developer meeting (see below).

## 22.3 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.<sup>2</sup> Note that the Java package structure does *not* directly correspond to the Maven/Git layout.

### 22.3.1 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 22.3.2.
- `org.matsim.core.*`, see Section 22.3.2.

<sup>1</sup>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.

<sup>2</sup> 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 10.4.
- `org.matsim.facilities.*`, see Section 10.5.
- `org.matsim.households.*`, see Section 10.6.
- `org.matsim.jaxb.*`, containing automatically or semi-automatically generated adapter classes to read XML files using Java Architecture for XML Binding (JAXB).
- `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 12.
- `org.matsim.run`, see Section 22.3.2.
- `org.matsim.utils.*` containing various utilities such as the much-used `ObjectAttributes` (see Section 21.4.1).
- `org.matsim.vehicles.*`, see Section 10.7.
- `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 5).
- `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.

### 22.3.2 Core

The core is part of the main distribution (see the previous Section 22.3.1) 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.

### 22.3.3 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.<sup>3</sup> 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:

<sup>3</sup> It is currently at GitHub under <https://github.com/matsim-org/matsim-libs/tree/master/contribs>.

- accessibility, presented in Chapter ??.
- analysis, presented in Chapter 14.
- 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 18.
- emissions, presented in Chapter 15.
- 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 20.
- matrixbasedptrouter, presented in Chapter ??.
- matsim4urbansim, presented in Chapter ??.
- minibus, presented in Chapter 19.
- multimodal, presented in Chapter ??.
- networkEditor, presented in Chapter ??.
- otfvis, presented in Chapter 7.
- parking, presented in Chapter ??.
- roadpricing, presented in Chapter ??.
- socnetgen, presented in Chapter ??.
- socnetsim, presented in Chapter ??.
- transEnergySim, presented in Chapter ??.
- wagonSim, presented in Chapter ??.

#### 22.3.4 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 21 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.

### 22.3.5 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,<sup>4</sup>
- 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>.

### 22.3.6 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 22.2 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.

## 22.4 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 Schweizerischer Nationalfonds (SNF) or Deutsche Forschungsgemeinschaft (DFG)—via projects for governmental entities and projects where science and industry contribute equally—e.g., Commission for Technology and Innovation (CTI) 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

---

<sup>4</sup>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.

## 22.5 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 under <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/maillinglists>. 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.

## 22.6 Your Contribution to MATSim

The technical details, i.e., the MATSim extension points, on where to hook with MATSim are detailed in Chapter 21. Here, the different ways of contributing to MATSim according to the roles presented in Section 22.2 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. 98
- API** Application Programming Interface. 9, 27, 63, 64, 159, 168, 174, 176–178, 185
- ARTEMIS** Assessment and Reliability of Transport Emission Models and Inventory Systems, a harmonized emission model in the European Union (EU), possibly a predecessor of HBEFA, version 3.1. 124
- BVG** Berliner Verkehrsbetriebe (started as Berliner Verkehrsaktien-Gesellschaft). 146
- CAS** Complex Adaptive Systems. 3
- CSV** Comma-Separated Values. 53, 112
- CTI** Commission for Technology and Innovation. 177
- CUDA** Compute Unified Device Architecture, a parallel computing platform and API by NVIDIA. 168
- DEQSim** Discrete Event Queue Simulation. 6, 7, 37
- DFG** Deutsche Forschungsgemeinschaft. 177
- DRT** Demand Responsive Transport. 143
- DTD** Document Type Description. 68, 81
- DVRP** Dynamic Vehicle Routing Problem. 136–138, 141–143
- EMME** Equilibre Multimodal Multimodal Equilibrium. See <http://www.inrosoftware.com/en/products/emme/>. 28, 69
- EMT** Emission Modeling Tool developed by Hülsmann et al. (2011) and Kickhöfer et al. (2013), see Chapter 15. 123, 124, 126
- EPSP** European Petroleum Survey Group. 21
- ESRI** Environmental Systems Research Institute. 49
- ETH** Eidgenössische Technische Hochschule. 45, 171
- EU** European Union. 185
- FCL** Future Cities Laboratory. 171
- FIFO** First In, First Out. 87, 142

- GIS** Geographic Information System. 21, 113
- GPLv2** GNU General Public License version 2.0. 172
- GPS** Global Positioning System. 19, 20, 49
- GTFS** General Transit Feed Specification. 95
- GUI** Graphical User Interface. 10, 12, 22, 26
- HBefa** Handbook on Emission Factors for Road Transport, version 3.1. See <http://www.hbefa.net>. 123–127, 185
- HPCC** High-Performance Computing Clusters. 35
- ICT** Information and Communications Technology. 135
- IDE** Integrated Development Environment. 10, 27, 33, 178
- ILS** Institut für Land- und Seeverkehr – Institute for Land and Sea Transport Systems. 171
- IVT** Institut für Verkehrsplanung und Transportsysteme – Institute for Transport Planning and Systems. 171
- JAR** Java ARchive. 10
- Java SE** Java Standard Edition. 9
- JAXB** Java Architecture for XML Binding. 175
- JDEQSim** Java Discrete Event Queue Simulation. 6, 7, 35, 37
- JOGL** Java OpenGL. 63
- JOSM** Java Open Street Map Editor. 25
- JSON** JavaScript Object Notation. 116
- MATSim** Multi-Agent Transport Simulation. See <http://www.matsim.org>. 3–7, 9–23, 25–29, 33–40, 44, 45, 47, 49, 52, 55, 57–59, 63, 68, 69, 74, 76–80, 84, 85, 87–91, 94, 95, 97, 98, 101, 102, 105–107, 109, 121, 123–127, 136, 138–141, 145, 146, 149–152, 154, 155, 158, 159, 161–166, 168–172, 174–179, 189
- MIMOSA** Modélisation Isentrope du transport Méso-échelle de l’Ozone Stratosphérique par Advection. 124
- MNL** Multinomial Logit Model. 40, 153
- mobsim** Mobility simulation. Also, depending on the context and specific mobility simulation capabilities, called network loading, traffic flow simulation or synthetic reality. 5, 6, 12, 14, 16, 18, 22, 34–36, 58–62, 64, 65, 102, 106, 154, 159–163, 167–169, 175, 190
- MOVES** Motor Vehicle Emission Simulator. See <http://www.epa.gov/otaq/models/moves/>. 124
- MSA** Method of Successive Averages. See, e.g., Liu et al. (2007). 41, 106
- MVI** OTFVis Movie File, not to be confused with the “Musical Video Interactive” file usually abbreviated mvi. 57–63
- O-D** Origin-Destination. 49, 167
- OpenGL** Open Graphics Library. 63, 64

- OS** Operating System. 59, 63
- OSM** OpenStreetMap (OpenStreetMap, 2015). 19, 45, 69, 77, 190
- OTFVis** On The Fly Visualizer. 10, 14, 27, 57–65
- PCU** Passenger Car Unit. 94
- PhD** Philosophiae Doctor – Doctor of Philosophy. 171, 172
- PHEM** Passenger Car and Heavy-duty Emission Model. 124
- PT** Public Transport. 91, 94, 95
- QGIS** Quantum GIS. 21, 28
- RAM** Random Access Memory. 59
- Scala** SCALable LAnguage. See <http://www.scala-lang.org/>. 28
- SI** Système International (d’Unités): International System (of Units). 75
- SNF** Schweizerischer Nationalfonds. 177
- SPI** Service Provider Interface. 161
- SUMO** Simulation of Urban Mobility. See <http://www.dlr.de/ts/sumo/en/>. 28
- TRANSIMS** TRansportation ANalysis and SIMulation System. See <https://code.google.com/archive/p/transims/>. 3
- TU** Technische Universität. 28, 171
- URL** Uniform Resource Locator. 22
- UTM** Universal Transverse Mercator. 19, 21
- VISSIM** Verkehr In Städten – SimulationsModell. See <http://www.ptv.de>. 124
- VISUM** Verkehr In Städten – Umliegung. See <http://www.ptv.de>. 28, 69, 90, 175
- VPL** VerkehrsPLANung. See <http://www.ivt.ethz.ch/vpl/>. 171
- VRP** Vehicle Routing Problem. 136, 137
- VSP** VerkehrsSystemPlanung und Verkehrstelematik – The Transport Systems Planning and Transport Telematics group at TU Berlin. See <https://www.vsp.tu-berlin.de>. 171
- VTTS** Value of Travel Time Savings. 102, 104
- WKT** Well-Known Text. 21
- XML** Extensible Markup Language, see <http://www.w3.org/XML/>. 15, 33, 68, 74, 76, 116, 126, 169, 175
- YAML** Yet Another Markup Language. 54



# 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.. 170

**Activity** The central element of modern activity-based modeling (see below). 4, 48, 191

**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. 190

**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, 105

**C++** An object-oriented programming language with full control of memory management. 6

**command line** A method to give input to computers.. 12, 93

**Configuration file** The main configuration screw for MATSim, often just referred to as config file or as config.xml. Also see config. 10–15, 17, 21–23, 26, 33–41, 58–60, 75–78, 80, 87, 94, 169, 174, 177, 189

**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. 11, 21, 33, 35, 38, 53, 108, 160, 164, 189

**Contribution** An extension contributed by the MATSim community and hosted in the MATSim repository. See <http://matsim.org/extensions..> 27, 58, 121, 136, 175, 177, 178

**Eclipse** The standard integrated development environment (IDE) used by the MATSim developers. 10, 178

**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. 18, 160, 162, 163, 167–169

**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 21. 27, 161, 170, 174, 177, 189

**Facility** An optional element in MATSim to further specify an activity location. 48, 77, 189

**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, 167, 168

**Git** A free and open source distributed version control system. 174, 178

**GitHub** A web-based Git repository hosting service, see <https://github.com/>. 10, 174, 175, 177

**Google Earth** Google's virtual globe. 21

**Identifier** A name that labels an object in a unique way. 20, 83, 85, 161

**Iteration** Numerical equilibrium search methods, such as MATSim, are iterative. A MATSim run is thus composed of a configurable number of iterations. 17, 190

**Java** A modern, object-oriented, cross-platform programming language run in virtual machines. 4, 10, 26–28, 35, 37, 45, 58, 63, 68, 74, 86, 94, 161, 165, 168–170, 174, 175, 189, 190

**Javadoc** Source code documentation compiled from javadoc annotations in the source files. 158, 178

**Jenkins** A software tool for continuous integration. 178

**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, 167, 191

**Link** A network component representing streets. 15

**MATSim run** A configurable number set of iterations, typically ending with an equilibrium solution of transport supply and demand. 4, 10, 17, 34, 169, 190

**Maven** A build automation tool by Apache tailored to Java. 10, 26, 27, 121, 159, 170, 174, 177, 178

**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. 47, 77

**Multimodal** Combining different means of transport. 28, 36, 88, 89, 145

**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. 15

**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. 149, 150, 191

**Osmosis** Command line Java application for processing OSM data. See <http://wiki.openstreetmap.org/wiki/Osmosis>. 69

- Plan** The agent's day schedule and, after run completion, an associated score. 4, 16, 82, 162
- QSim** The standard MATSim mobsim. 6, 7, 14, 34–37, 79, 82–87, 90, 160
- Replanning** The stage when agents modify their plans. 5, 12, 16, 38, 159, 162, 169
- 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, 160, 168
- 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, 98, 169, 190
- Scoring** see score. 37, 169
- Senozon AG** A spin-off company founded by two core developers of MATSim. 45, 171, 177
- Simunto GmbH** A company founded by a core developer of MATSim. 44, 45, 172, 173, 177, 191
- SimWrapper** An open-source web-based data visualization platform. 52, 54, 55, 120
- Stage** A stage is part of a trip, performed with a single mode. In MATSim called leg. 190
- 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, 82, 85, 88, 89
- Teleported** see teleportation. 89
- Trip** The connection between two activities, composed of multiple legs. 18, 190
- Utility** A central economic concept representing satisfaction through goods consumption. The MATSim score can be interpreted in utility units. 98, 191
- Via** Visualization and analysis tool for MATSim by Simunto GmbH. 10, 14, 25, 175





## 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.
- Ahn, K. and H. Rakha (2008) The effects of route choice decisions on vehicle energy consumption and emissions, *Transportation Research Part D: Transport and Environment*, **13** (3) 151–167.
- André, M. and M. Rapone (2009) Analysis and modelling of the pollutant emissions from European cars regarding the driving characteristics and test cycles, *Atmospheric Environment*, **43** (5) 986–995.
- 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/>.
- Axhausen, K. W. (2006) Neue Modellansätze der Verkehrsnachfragesimulation: Entwicklungslinien, Stand der Forschung, *Forschungsperspektiven, Stadt Region Land*, **81**, 149–164.
- Baker, J., D. Grewel and A. Parasuraman (1994) The influence of store environment on quality inferences and store image, *Journal of the Academy of Marketing Science*, **22** (4) 328–339.
- 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.
- Beckx, C., T. A. Arentze, L. Int Panis, D. Janssens, J. Vankerkorn and G. Wets (2009) An integrated activity-based modelling framework to assess vehicle emissions: Approach and application, *Environment and Planning B*, **36** (6) 1086–1102.
- Ben-Akiva, M. E. and B. Boccara (1995) Discrete choice models with latent choice sets, *International Journal of Research in Marketing*, **12** (1) 9–24.
- 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.

- Burnett, K. P. (1980) Spatial constraints-oriented modelling as an alternative approach to movement: Microeconomic theory and urban policy, *Urban Geography*, 1 (1) 53–67.
- Burnett, K. P. and S. Hanson (1979) Rationale for an alternative mathematical approach to movement as complex behavior, *Transportation Research Record*, 723, 11–24.
- 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.
- Charlton, W. and B. Sana (2022) Simwrapper, an open source web-based platform for interactive visualization of microsimulation outputs and transport data (submitted version), *Procedia Computer Science*. The 14th International Conference on Ambient Systems, Networks and Technologies (ANT/ABMTRANS) / Affiliated Workshops.
- Charypar, D. (2008) Efficient algorithms for the microsimulation 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 <http://vsp.berlin/publications>.
- 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](https://doi.org/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](https://doi.org/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.
- Creutzig, F. and D. He (2009) Climate change mitigation and co-benefits of feasible transport demand policies in Beijing, *Transportation Research Part D: Transport and Environment*, 14 (2) 120–131.
- 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.

- Eroglu, S. and G. D. Harrell (1986) Retail crowding: Theoretical and strategic implications, *Journal of Retailing*, **62** (4) 346–363.
- Eroglu, S. and K. A. Machleit (1990) An empirical study of retail crowding: Antecedents and consequences, *Journal of Retailing*, **66** (2) 201–221.
- Eroglu, S., K. A. Machleit and T. Feldman Barr (2005) Perceived retail crowding and shopping satisfaction: The role of shopping values, *Journal of Business Research*, **58** (8) 1146–1153.
- 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/tmp/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.
- Frejinger, E., M. Bierlaire and M. E. Ben-Akiva (2009) Sampling of alternatives for route choice modeling, *Transportation Research Part B: Methodological*, **43** (10) 984–994.
- 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 <http://vsp.berlin/publications>.
- 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.
- Harrell, G. D., M. D. Hutt and J. C. Anderson (1980) Path analysis of buyer behavior under conditions of crowding, *Journal of Marketing Research*, **17** (1) 45–51.
- Hatzopoulou, M. and E. J. Miller (2010) Linking an activity-based travel demand model with traffic emission and dispersion models: Transport’s contribution to air pollution in Toronto, *Transportation Research Part D: Transport and Environment*, **15** (6) 315–325.
- Hirschmann, K., M. Zallinger, M. Fellendorf and S. Hausberger (2010) A new method to calculate emissions with simulated traffic conditions, in: *Intelligent Transportation Systems Conference (ITSC)*, Madeira, September 2010.
- Horni, A. (2013) Destination choice modeling of discretionary activities in transport microsimulations, Ph.D. Thesis, ETH Zurich, Zurich.
- Horni, A. (2016) MATSim destination choice documentation, webpage, <http://matsim.org/docs/extensions/destination-choice>.
- Horni, A. and K. W. Axhausen (2012) MATSim agent heterogeneity and week scenario, *Working Paper*, **836**, IVT, ETH Zurich, Zurich.

- Horni, A., D. Charypar and K. W. Axhausen (2011a) Variability in transport microsimulations investigated with the multi-agent transport simulation MATSim, *Working Paper*, 692, IVT, ETH Zurich, Zurich.
- Horni, A., F. Ciari and K. W. Axhausen (2012a) Coupling customers' destination choice and retailers' location choice in MATSim, *Working Paper*, 808, IVT, ETH Zurich, Zurich.
- Horni, A. and D. Grether (2007) Counts, internal presentation, MATSim-T Workshop, IVT, ETH Zurich, Castasegna, October 2007.
- Horni, A., L. Montini and K. W. Axhausen (2013) An Agent-Based Cellular Automaton Cruising-For-Parking Simulation, *IATBR Special Issue of Transportation Letters*.
- Horni, A., K. Nagel and K. W. Axhausen (2012b) High-resolution destination choice in agent-based models, *Annual Meeting Preprint*, 12-1988, Transportation Research Board, Washington, D.C. Also VSP WP 11-17 <http://vsp.berlin/publications>.
- 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 (2011b) The Zurich scenario: A technical overview, *Working Paper*, 687, IVT, ETH Zurich, Zurich.
- Hui, M. K. and J. E. G. Bateson (1991) Perceived control and the effects of crowding and consumer choice on the service experience, *Journal of Consumer Research*, 18 (2) 174-184.
- Hülsmann, F., R. Gerike, B. Kickhöfer, K. Nagel and R. Luz (2011) Towards a multi-agent based modeling approach for air pollutants in urban regions, in: *Conference on "Luftqualität an Straßen"*, 144-166. Also VSP WP 10-15 <http://vsp.berlin/publications>.
- Hülsmann, F., B. Kickhöfer and R. Gerike (2013) Air pollution hotspots in urban areas – how effective are pricing strategies to comply with the EU limits for  $NO_2$ ?, in: R. Gerike, K. Roller and F. Hülsmann (eds.) *Strategies for Sustainable Mobilities: Opportunities and Challenges*, 105-128, Ashgate Publishing Ltd.
- Hörl, S. and M. Balac (2021) Introducing the eqasim pipeline: From raw data to agent-based transport simulation, *Procedia Computer Science*, 184, 712-719, [doi:10.1016/j.procs.2021.03.089](https://doi.org/10.1016/j.procs.2021.03.089). ABMTRANS '21.
- 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](https://doi.org/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.
- Kaddoura, I., L. Kröger and K. Nagel (2017) An activity-based and dynamic approach to calculate road traffic noise damages, *Transportation Research Part D: Transport and Environment*, 54, 335-347, [doi:10.1016/j.trd.2017.06.005](https://doi.org/10.1016/j.trd.2017.06.005).
- Kickhöfer, B., F. H. Hülsmann, R. Gerike and K. Nagel (2013) Rising car user costs: Comparing aggregated and geo-spatial impacts on travel demand and air pollutant emissions, in: T. Vanoutrive and A. Verhetsel (eds.) *Smart Transport Networks: Decision Making, Sustainability and Market structure*, NECTAR Series on Transportation and Communications Networks Research, 180-207, Edward Elgar Publishing Ltd, Cheltenham.
- Kickhöfer, B. and J. Kern (2015) Pricing local emission exposure of road traffic: An agent-based approach, *Transportation Research Part D: Transport and Environment*, 37 (1) 14-28, [doi:10.1016/j.trd.2015.04.019](https://doi.org/10.1016/j.trd.2015.04.019).
- Kickhöfer, B. and K. Nagel (2011) Mapping emissions to individuals – new insights with multi-agent transport simulations, in: *12th Conference on Computers in Urban Planning and Urban Management (CUPUM)*, Lake Louise, Canada. Also VSP WP 11-02 <http://vsp.berlin/publications>.

- Kickhöfer, B. and K. Nagel (2016) Towards high-resolution first-best air pollution tolls, *Networks and Spatial Economics*, **16** (1) 175–198, doi:10.1007/s11067-013-9204-8.
- Kickhöfer, B. (2014) Economic policy appraisal and heterogeneous users, Ph.D. Thesis, doi:10.14279/depositonce-4089.
- Knight, F. H. (1924) Some fallacies in the interpretation of social cost, *Quarterly Journal of Economics*, **38** (4) 582–606.
- Kraschl-Hirschmann, K., M. Zallinger, R. Luz, M. Fellendorf and S. Hausberger (2011) A method for emission estimation for microscopic traffic flow simulation, in: *IEEE Forum on Integrated and Sustainable Transportation Systems*, Vienna, June 2011.
- 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 <http://vsp.berlin/publications>.
- 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 <http://vsp.berlin/publications>.
- 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.
- Manski, C. F. (1977) The structure of random utility models, *Theory and Decision*, **8** (3) 229–254.
- MATSim (2016) Multi-Agent Transportation Simulation, webpage, <http://www.matsim.org>.
- Mayobre, M. (2018) Erweiterung einer Verkehrssimulationssoftware um ein Modell zur Berechnung von Unfallkosten, Master's thesis, TU Berlin, Institute for Land and Sea Transport Systems, Berlin.
- 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.

- Michalewicz, Z. and D. B. Fogel (2004) *How to Solve It: Modern Heuristics*, Springer, Heidelberg.
- Michiels, H., I. Mayers, L. Int Pais, L. De Nocker, F. Deutsch and W. Lefebvre (2012)  $PM_{2.5}$  and  $NO_x$  from traffic – human health impacts, external costs and policy implications from the Belgian perspective, *Transportation Research Part D: Transport and Environment*, **17** (8) 569–577.
- 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 <http://vsp.berlin/publications>.
- Neumann, A. (2014) A paratransit-inspired evolutionary process for public transit network design, Ph.D. Thesis, TU Berlin, Berlin.
- Neumann, A., D. Röder and J. W. Joubert (2015) Towards a simulation of minibuses in South Africa, *Journal of Transport and Land Use*, **8** (1) 137–154, doi:10.5198/jtlu.2015.390.
- OpenStreetMap (2015) The Free Wiki World Map, webpage, <http://www.openstreetmap.org>.
- Pagliara, F. and H. J. P. Timmermans (2009) Choice set generation in spatial contexts: A review, *Transportation Letters*, **1** (1) 181–196.
- 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.
- Pigou, A. C. (1920) *The Economics of Welfare*, Macmillan and Co., London.
- Pons, F., M. Laroche and M. Mourali (2006) Consumer reactions to crowded retail settings: Cross-cultural differences between North America and the Middle East, *Psychology & Marketing*, **23** (7) 555–572.
- 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.
- Schüssler, N. (2010) Accounting for similarities between alternatives in discrete choice models based on high-resolution observations of transport behaviour, Ph.D. Thesis, ETH Zurich, Zurich.
- 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.
- Simunto GmbH (2018) Via – visualization and analysis tool, webpage, <https://www.simunto.com/via>.
- 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.
- Song, G., L. Yu and Y. Zhang (2012) Applicability of traffic microsimulation models in vehicle emissions estimates, *Transportation Research Record*, **2270**, 132–141.
- 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](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.
- Thill, J.-C. (1992) Choice set formation for destination choice modelling, *Progress in Human Geography*, **16** (3) 361–382.
- 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>.
- U.S. Bureau of Public Roads (1964) *Traffic Assignment Manual*, U.S. Department of Commerce, Washington, D.C.
- Vickrey, W. S. (1969) Congestion theory and transport investment, *The American Economic Review*, **59** (2) 251–260.
- Vovsha, P., E. Petersen and R. Donnelly (2002) Microsimulation in travel demand modeling: Lessons learned from the New York best practice model, *Transportation Research Record*, **1805**, 68–77.
- 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.
- Wardrop, J. G. (1952) Some theoretical aspects of road traffic research, *Proceedings of the Institution of Civil Engineers*, **1** (3) 325–362.
- Weilenmann, M., J.-Y. Favez and R. Alvarez (2009) Cold-start emissions of modern passenger cars at different low ambient temperatures and their evolution over vehicle legislation categories, *Atmospheric Environment*, **43** (15) 2419–2429.
- Wismans, L., R. van den Brink, L. Brederode, K. Zantema and E. van Berkum (2013) Comparison of estimation of emissions based on static and dynamic traffic assignment models, in: *92nd Annual Meeting of the Transportation Research Board*, Washington, D.C., January 2013.
- Zheng, J. and J. Y. Guo (2008) Destination choice model incorporating choice set formation, in: *87th Annual Meeting of the Transportation Research Board*, Washington, D.C., January 2008.

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](https://doi.org/10.1016/j.procs.2015.05.017).