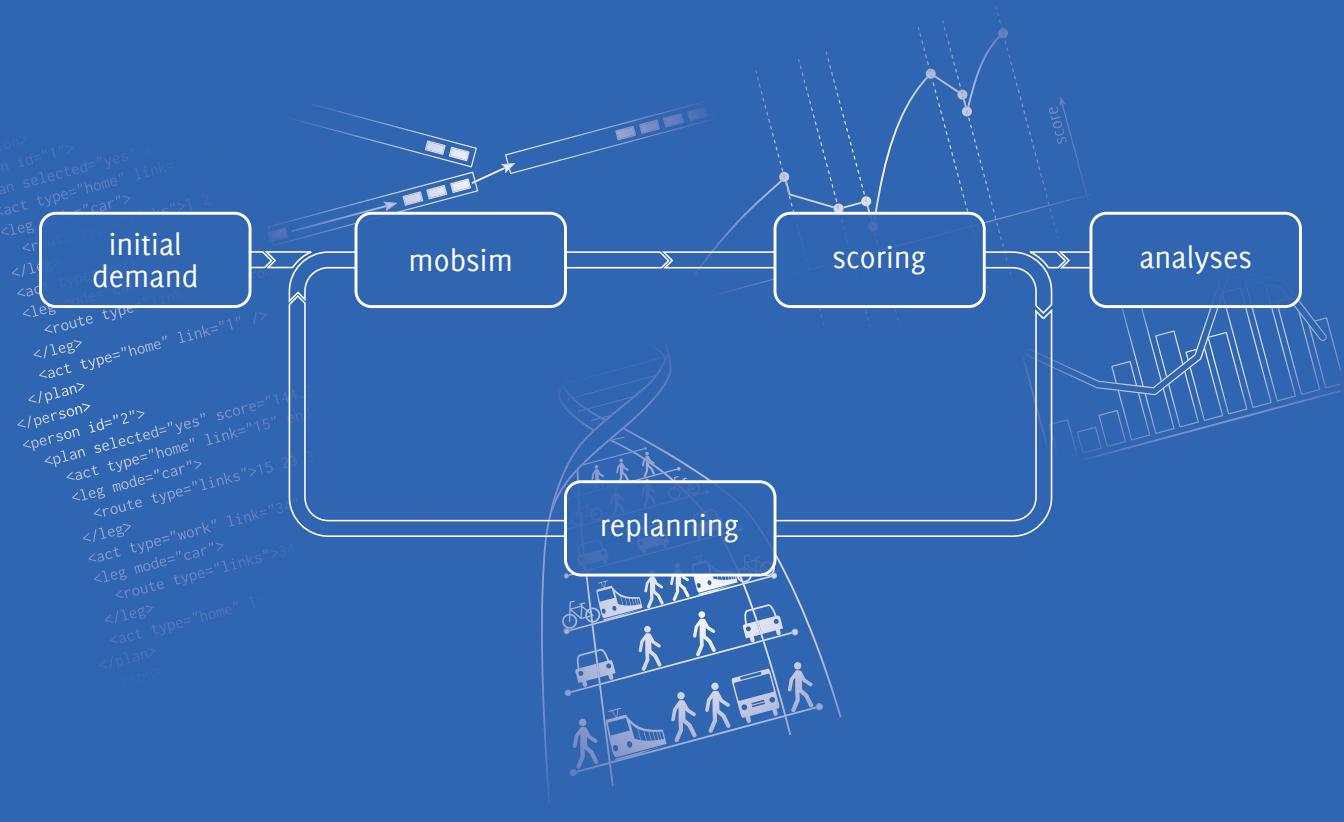
The Multi-Agent Transport Simulation MATSim

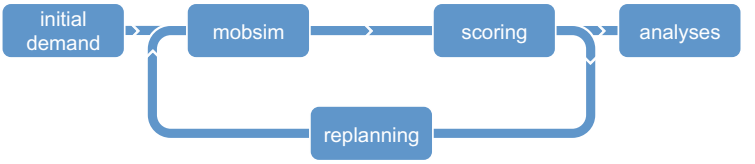
edited by
Andreas Horni, Kai Nagel, Kay W. Axhausen



MATSim
Multi-Agent Transport Simulation

The Multi-Agent Transport Simulation MATSim

Edited by
Andreas Horni, Kai Nagel, Kay W. Axhausen



]u[

ubiquity press
London

Published by
Ubiquity Press Ltd.
6 Windmill Street
London W1T 2JB
www.ubiquitypress.com

First published 2016

Cover Illustration by Dr. Marcel Rieser, Senozon AG

Print and digital versions typeset by diacriTech.

ISBN (Hardback): 978-1-909188-75-4
ISBN (PDF): 978-1-909188-76-1
ISBN (EPUB): 978-1-909188-77-8
ISBN (Mobi/Kindle): 978-1-909188-78-5

DOI: http://dx.doi.org/10.5334/baw

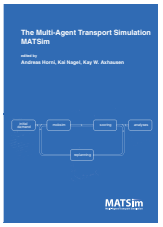
The full text of this book has been peer-reviewed to ensure high academic standards. For full review policies, see http://www.ubiquitypress.com/

Suggested citation:
Horni, A, Nagel, K and Axhausen, K W (eds.) 2016 *The Multi-Agent Transport Simulation MATSim*. London: Ubiquity Press. DOI: http://dx.doi.org/10.5334/baw.
License: CC-BY 4.0

Contents

Cover and Title Photos

The following cover and title photos have been provided by Dr. Marcel Rieser, Senozon AG.



©Dr. Marcel Rieser, Senozon AG

Portland, Oregon. View from the south to the city center, from the Portland Aerial Tram. June 2008. ©Dr. Marcel Rieser, Senozon AG

Zürich, Switzerland. Tracks at Zürich Main Station. May 2011. ©Dr. Marcel Rieser, Senozon AG

Berne, Switzerland. Car and bike park at Berne Main Station. June 2011. ©Dr. Marcel Rieser, Senozon AG

Gotthard railway model at the Swiss Museum of Transport, Lucerne, Switzerland. February 2004. ©Dr. Marcel Rieser, Senozon AG

Preface

Developing complex software for over a decade with a heterogeneous group of engineers and scientists, each with widely different skill levels and expertise across multiple locations around the world, requires dedication and mechanisms unusual for a university environment.

This book is one of these mechanisms. It allows us, collectively, to take stock and present a coherent state-of-the-system: for us and anyone interested in this approach. It highlights basics for the student who wants to undertake a small first research project as part of his or her degree, provides a description of the main functionalities, in detail, for the engineer setting up MATSim (Multi-Agent Transport Simulation) to conduct a policy analysis and, finally, fits the approach into the theoretical background of complex systems in computer science and physics.

The choice of the additional e-book format is an advantage, as it allows us to keep the book up-to-date with future chapters, revisions and, if necessary, errata. Equally importantly it allows you, the readers, to select those sections relevant to your needs.

The book comes at an important time for the system; for most of the first decade, its use was limited to the original developers and users in Berlin and Zürich. It is now much more widely consulted around the world, as we document in the chapter summarizing contributions on scenarios so far.

Scenario: This term will occur again and again. In MATSim context, it 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. Within these scenarios, the user can experiment and explore with behavioral utility function parameters, with the sampling rate of the population between 1 % and 100 %, with algorithm parameters, e.g., the share of the sample engaged in replanning in any iteration, or behavioral dimensions or exact settings necessary to avoid gridlock due to the traffic flow dynamics. The creation of a scenario is a substantial effort, and the framework makes a number of tools available to accelerate it: population synthesizers, network editors, network converters between popular formats and the MATSim representation, e.g., OSM (OpenStreetMap) or GTFS (General Transit Feed Specification), semi-automatic network matching to join information, among others.

A large group of colleagues has been involved and many of them are contributors to this book; this is a list of those involved, other than ourselves, in Berlin, Singapore and Zürich.

Amit Agarwal
Milos Balac
Dr. Michael Balmer
Henrik Becker
Joschka Bischoff
Patrick Bösch
Dr. David Charypar
Dr. Nurhan Cetin
Dr. Artem Chakirov
Dr. Yu Chen
Dr. Francesco Ciari
Dr. Christoph Dobler
Thibaut Dubernet
Dr. Alexander Erath
Dr. Matthias Feil
Prof. Dr. Gunnar Flötteröd
Pieter J. Fourie
Dr. Christian Gloor
Dr. Dominik Grether
Dr. Jeremy K. Hackney
Dr. Johannes Illenberger
Prof. Dr. Johan W. Joubert
Ihab Kaddoura
Dr. Benjamin Kickhöfer
Dr. Gregor Lämmel
Nicolas Lefebvre
Dr. Michal Maciejewski
Dr. Fabrice Marchal
Alejandro Marmolejo
Dr. Konrad Meister
Dr. Manuel Moyo Oliveros
Kirill Müller
Dr. Andreas Neumann
Dr. Thomas Nicolai
Sergio A. Ordóñez Medina
Dr. Bryan Raney
Dr. Marcel Rieser
Dr. Nadine Rieser-Schüssler
Daniel Röder
Mohit Shah
Dr. Lijun Sun
Alexander Stahel
Prof. Dr. David Strippgen
Theresa Thunig
Dr. Basil Vitins
Michael Van Eggermond
Dr. Rashid Waraich
Dominik Ziemke
Michael Zilske

Additional contributors are mentioned as authors of their respective chapters in this book. We hope to acknowledge the contributions of more colleagues from other groups in future versions of this book and in the software.

Special thanks go to a number of people who greatly helped improving this book beyond their own chapters. Benjamin Kickhöfer's deep knowledge of MATSim's mathematical base, particularly its interpretation within the discrete choice framework, made the discussions accompanying the writing of this book very fruitful. Thibaut Dubernet's, Marcel Rieser's and Michael Zilske's outstanding expertise on software core development helped us very much and also improved the software structure during the writing of this book. Marcel Rieser's layout and illustrations greatly improved the book's appearance. Joschka Bischoff's effort to document basic information about every module will greatly help readers make a quick step into respective functionality.

The efficient and productive copy editing by Karen Ettlin is gratefully acknowledged.

The reported effort was funded and supported over the years by numerous agencies. Several particularly important sources are: ETH (Eidgenössische Technische Hochschule) Zürich and TU (Technische Universität) Berlin, the DFG (Deutsche Forschungsgemeinschaft), the SNF (Schweizerischer Nationalfonds), the Swiss ASTRA (BundesAmt für STRAssen), and the NRF (Singaporean National Research Foundation), through their repeated grants and projects supporting different dissertations over the years. A more complete list is provided on pages xxi ff. This support is gratefully acknowledged by all researchers.

The publication of this book was funded by the following institutions. The publisher services are funded by the EU (European Union) FP7 post-grant Open Access Pilot (OpenAIRE) and by DFG. The book's copy-editing is funded by the SNF under B-0010_166808. The support is highly appreciated.

We hope this book captures the interest of more researchers and engineers and encourages them to get involved in this joint effort. This would enable us to provide this framework, which has to be continuously adapted to our policy needs, together and ensure that it stays at the forefront of travel behavior modeling.

The editors
Andreas Horni, Kai Nagel, Kay W. Axhausen
Zürich and Berlin, February 2016

Acknowledgments

A project this dispersed and as long as the MATSim (Multi-Agent Transport Simulation) project draws on many sources for its support. We hope that we have not forgotten any institution here. We are grateful to all of them that they have made this open-source effort possible and we hope that they will continue to do so in the spirit of intellectual discovery and sharing.

In every case, we have to thank our home institutions for providing the basic intellectual and computing infrastructure for our work. ETH (Eidgenössische Technische Hochschule) Zürich was home to Prof. Nagel and his group when he started the project and continues to be the basis for Prof. Axhausen and his team. TU (Technische Universität) Berlin became Prof. Nagel's new platform after his move. Both institutions provided support through base funding for staff, servers and data access, which allow us to provide ongoing support to the overall project.

The following projects and sponsors funded particular persons and implementations:

TU Berlin (Kai Nagel, Amit Agarwal, Ulrike Beuck, Joschka Bischoff, Yu Chen, Gunnar Flötteröd, Dominik Grether, Johannes Illenberger, Ihab Kaddoura, Benjamin Kickhöfer, Gregor Lämmel, Michal Maciejewski, Manuel Moyo Oliveros, Andreas Neumann, Thomas Nicolai, Marcel Rieser, David Strippgen, Theresa Thunig, Jakub Wilk, Dominik Ziemke, Michael Zilske) undertook this work in the framework of the following projects: "COOPERS (Co-Operative Networks for Intelligent Road Safety) (EU (European Union) 026814); "Modelling and simulation approaches for livable cities" (Volvo Research and Education Foundation SP-2004-49); "Travel impacts of social networks and networking tools" (Volkswagen Stiftung I/82 714); "Numerical Last-mile Tsunami Early Warning and Evacuation Information System" (BMBF (Bundesministerium für Bildung und Forschung/Federal Ministry of Education and Research) 03FG0666E); "Adaptive Traffic Control" (BMBF 03NAPAI4); "State Estimation for traffic simulations as coarse grained systems" (DFG (Deutsche Forschungsgemeinschaft) NA 682/1-1); "Detailed assessment of transport measures using micro-simulation" (DFG NA 682/3-1); "Simulation of Multidestination Pedestrian Crowds" (DFG NA 682/5-1); "SustainCity: Micro-simulation for the prospective of sustainable cities in Europe (EU 7th Framework 244557); "Contributions of transport towards the realization of a 2000 W city" (DFG NA 682/6-1); "GRIPS (GIS-based Risk analysis, Information, and Planning System for the evacuation of areas)" (BMBF 13N11382); "MINTE (MItigating Negative Transport Externalities in industrialized and newly industrializing

countries)" (DAAD (Deutscher Akademischer Austauschdienst – German Academic Exchange Service) scholarship for doctoral students), "eCab: Simulation-based system for the sustainable management of electrically powered taxi fleets" (Einstein Stiftung Berlin A-2012-132); "Optimization and network wide analysis of traffic signal control" (DFG NA 682/7-1); "MAXess: Measuring accessibilities for policy evaluation" (ERA (European Research Action – Country consortia), ERAfrica, BMBF 01DG14008); "An agent-based evolutionary approach for the user-oriented optimization of complex public transit systems" (DFG NA682/11-1).

ETH Zürich (Kay Axhausen, Milos Balac, David Charypar, Francesco Ciari, Christoph Dobler, Thibout Dubernet, Andreas Horni, Nadine Rieser, Rashid Waraich) could also draw on the following grants: "A generalized approach to population synthesis" (SNF (Schweizerischer Nationalfonds) 205121_138270 25); "Agent-based modelling of retailers and their reactions to road pricing" (ETH TH-19042); "Agent-based simulation for location-based services" (KTI (Kommission für Technologie und Innovation) 8443.1 ESPP-ES); "An investigation of strategies leading to a 2000 W City using a bottom-up model of urban energy flows" (SNF 105218-122632 1); "Assessment of the impacts of the Westumfahrung Zürich (Kanton Zürich)"; "Autonomous Cars—The next revolution in mobility" (SNF 200021_159234 43); "Choice models for transport modelling: Accounting for similarities between alternatives in large scale choice sets" (SNF 205120-121889 14); "Deriving and assessing strategies for limiting the spread of airborne diseases using a social contact model: The case of influenza" (SNF); "Destination Choice Modeling for Discretionary Activities: Fundamentals of Choice Set Formation and Impacts of Spatial Competition" (SNF 205121_132086 20); "Dynamic Traffic Self-organization in China: Network Spatial-temporal Methodology and MATSim Simulation" (SNF IZ69Z0_13113917); "Integrated modelling and analysis of energy and transport systems" (ETH TH-22 07-03); "Large-scale multi-agent simulation of travel behaviour and traffic flow" (ETH TH-7959); "Large-scale stochastic optimization for agent-based traffic simulations" (ETH TH-18951); "MAXess: Measuring accessibility in policy evaluation" (ERA, ERAfrica IZEAZ0_154310 37); "Models without (personal) data?" (SNF 200021_144134 29); "Optimising public transport: Making smart cards more useful" (SNF IZKSZ2_162185 44); "Post Car World" (SNF CRSII1_147687 21); "SCCER (Swiss Competence Center for Energy Research) Energy and Mobility" (KTI 33290); "Sharing is Saving: how collaborative mobility can reduce the impact of energy consumption for transportation" (NFP (Nationales Forschungsprogramm) 407140_153807 41); "Simulation evacuation scenarios and Schwingerfest: Evacuation study" (BABS (Bundesamt für Bevölkerungsschutz, Switzerland)); "SURPRICE (Sustainable mobility through Road User Charging)" (ERA, ERA.net); "SustainCity: Micro-simulation for the prospective of sustainable cities in Europe" (EU 7th Framework 244557); "THELMA (Technology-centered ELectric Mobility Assessment)" (CCEM (Competence Center Energy and Mobility)); "ToPDAd (Tool supported Policy Development for regional Adaptation)" (EU 7th Framework 308620); "Travel behaviour in a dynamic spatial and social context: Modelling the Interdependence of Social Network Interactions and spatial choices" (SNF 105212-112482 10) and "Travel impacts of social networks and networking tools" (Volkswagen Stiftung I/82 714).

The NRF (Singaporean National Research Foundation) together with ETH Zürich supported the work of Alexander Erath, Pieter Fourie, Sergio Ordonez Medina, Artem Chakirov and Michael Van Eggermond as part of FCL (Future Cities Laboratory).

The co-operation which funded Lun Zhang's work (Tongji University) was based on two grants (EG01-032010, NIP02-092010) of the Sino-Swiss Cooperation Project Program funded by ETH Zürich.

The work reported by Senozon AG (Michael Balmer, Marcel Rieser, Daniel Röder, Christoph Dobler and Andreas Neumann) is based on projects undertaken since it was set up in 2010, especially noteworthy are the following clients: BVG (Berliner Verkehrsbetriebe), BfS (Bundesamt für Statistik – Federal Statistical Office), Peter Vovsha, Parsons Brinckerhoff, NY, Prof. Ulrich Weidmann, Transport Systems Group (VS) of the IVT (Institut für Verkehrsplanung und Transportsysteme – Institute for Transport Planning and Systems).

University of Pretoria (Johan Joubert) was supported by grants of the South African National Treasury and the National Research Foundation grant FA2007051100019.

At RMIT (Royal Melbourne Institute of Technology) Lin Padgham and Dhirendra Singh were supported by the ARC (Australian Research Council) Discovery DP1093290, ARC Linkage LP130100008 and Telematics Trust grants. They would like to thank Agent Oriented Software for the use of the JACK BDI (Belief Desire Intention) platform.

The work of Seungjae Lee and Atizaz Ali at the University of Seoul was supported by a grant (11 High-Tech Urban G06) from High-tech Urban Development Program funded by Ministry of Land, Infrastructure and Transport of Korean government.

At the National Institute for Environmental Studies, the research of Daisuke Murakami was supported by the Environment Research and Technology Development Fund (S-10) of Japan's Ministry of the Environment.

The work on the Trondheim scenario by Stefan Flügel, Julia Kern and Frederik Bockemühl was supported by the Research Council of Norway with "Future Sustainable Transport for Industry and Trade in Norway" (208420/F40).

The work on the Santiago de Chile scenario by Benjamin Kickhöfer and Alejandro Tirachini has been supported by Chile's CONICYT (Comisión Nacional de Investigación Científica y Tecnológica – National Commission for Scientific and Technological Research) through the FONDECYT (Fondo Nacional de Desarrollo Científico y Tecnológico) Grant 11130227.

The research presented by the University of Poznan (Michal Maciejewski, Waldemar Walerjanczyk) was partially supported by the grants PBS1/A6/11/2012 and ERA-NET-TRANSPORT-III/2/2014 from the National Centre for Research and Development (Poland).

At the Universite de Liege (Mario Cools, Jacques Teller, Ismail Saadi) the work was supported by the ARC grant for Concerted Research Actions, financed by the Wallonia-Brussels Federation on "Landuse change and future flood risk: influence of micro-scale spatial patterns (FLOODLAND)".

Oleg Saprykin, Olga Saprykina and Tatyana Mikheeva were supported by the Ministry of Education and Science of the Russian Federation at Samara State Aerospace University.

Chengxiang (Tony) Zhuge (Zhejiang University, Beijing Jiaotong University) and Chunfu Shao's project "Evolution Mechanism, Regulation and Control Methods of Urban Transportation Supply and Demand Structure" was funded by the National Natural Science Foundation of China (51338008).

Sashikanth Gurram, Abdul R. Pinjari and Amy L. Stuart work at the University of South Florida and benefited from a grant by the National Science Foundation (0846342) on "Tampa, Florida: High Resolution Simulation of Urban Travel and Network Performance for Estimating Mobile Source Emissions".

The work of Maxime Lenormand at UIB (Universitat Autónoma de Barcelona) and Miguel Picornell at Nommon was in the context of a EU 7th Framework grant (EUNOIA (Evolutive User-centric Networks fOr Intraurban Accessibility), 318367).

The work for Toronto (Adam Weis, Khandker Nurul Habib, Peter Kucirek, Eric Miller, CF Shao) was funded in part by an Natural Sciences and Engineering Research Council (Canada) Discovery Grant and by the sponsors of the University of Toronto Travel Modelling Group: Metrolinx, the Ontario Ministry of Transportation, the Cities of Toronto, Hamilton, Mississauga and Brampton, and the Regional Municipalities of Durham, Halton, Peel and York.

The work at Shinshu University (Rolando Armas) is supported by the Ecudoran National Secretariat of Higher Education, Science, Technology and Innovation.

National University of Ireland Maynooth and Dublin (Gavin McArdle, Aonghus Lawlor, Eoghan Furey) were supported by the Science Foundation Ireland by a Strategic Research Cluster grant (07/SRC/I1168) under the National Development Plan.

The work at the University of Melbourne (Nicole Roland) was based on an Australian Research Council grant on "Integrating Mobility on Demand" (Linkage Project LP120200130).

Daisuke Fukuda's work at Tokyo Tech was supported by a Grant-in-Aid for Scientific Research from the Japan Society for the Promotion of Science (B) number 25289160 and by the CART (Committee on Advanced Road Technology), Ministry of Land, Infrastructure, Transport, and Tourism, Japan.

The results from Erasmus University Rotterdam (Paul Bouman, Milan Lovric) were made possible by a grant of the NYBPM (Nederlandse Organisatie voor Wetenschappelijk Onderzoek – Netherlands Organization for Scientific Research) funding the ComPuTr (Complexity in Public Transport) project.

The research leading to the results reported by UCL (University College London) (Camilo Ruiz, Joan Serras, Mike Batty, Melanie Bosredon, Vassilis Zachariadis) has received funding from Engineering and Physical Sciences Research Council of UK (United Kingdom) under grant agreement number EP/G057737/1 (SCALE project; 2009–2013), the European Union 7th Framework Programme FP7/2007–2013 under grant agreement number 318367 (EUNOIA project) and the European Research Council under grant agreement number 249393 (MECHANICITY project; 2010–2015).

The past and ongoing work at KTH (Kungliga Tekniska Högskolan – Royal Institute of Technology) Stockholm (Gunnar Flötteröd) was based on the following grants: "IHOP2: Flexible coupling of disaggregate travel demand models and network simulation packages" (TRV (Trafikverket – Swedish Transport Administration) 2015/2950); "SMART-PT: Smart public Transport" (ERA, Eranet Transport III—Future traveling, VINNOVA 2014-03976) and "PETRA (PErsonal TRansport Advisor): an integrated platform of mobility patterns for Smart Cities to enable demand-adaptive transportation system" (EU 7th Framework Program 609042). He is supported by the KTH strategic research program in transport TRENoP (Transport REsearch with Novel Perspectives).

The data sources and support which the authors obtained are too numerous to list here. Please see the original papers, theses and reports as cited in the various chapters. Special thanks go to OSM (OpenStreetMap) and their contributors, who have made the procurement of high-quality highly detailed network data much easier than it was before.

Selected Sponsors

BABS	Bundesamt für Bevölkerungsschutz – Federal Office for Civil Protection	Switzerland
BMBF	Bundesministerium für Bildung und Forschung/Federal Ministry of Education and Research	Germany
DFG	Deutsche Forschungsgemeinschaft – German Research Foundation	Germany
ERA	European Research Action	Country consortia
EU	European Union	European countries
KTI	Kommission für Technologie und Innovation/ Commission for Technology and Innovation	Switzerland
NFP	Nationales Forschungsprogramm – National Research Program	Switzerland
NRF	National Research Foundation	Singapore
NSF	National Science Foundation	USA
SNF	Schweizerischer Nationalfonds – Swiss National Research Foundation	Switzerland

Contributors

Editors

Andreas Horni
Institute for Transport Planning and Systems (IVT)
ETH Zürich
horni@senozon.com

Kay W. Axhausen
Institute for Transport Planning and Systems (IVT)
ETH Zürich
axhausen@ivt.baug.ethz.ch

Kai Nagel
Transport Systems Planning and Transport Telematics (VSP)
TU Berlin
nagel@vsp.tu-berlin.de

Authors (alphabetically)

Amit Agarwal
Transport Systems Planning and Transport Telematics (VSP)
TU Berlin
amit.agarwal.iitd@gmail.com

Hernan Aguirre
Faculty of Engineering
Shinshu University, Japan
ahernan@shinshu-u.ac.jp

Atizaz Ali
Departement of Transportation Engineering
University of Seoul
atizaz.ali@uos.ac.kr

Yalcin Alver
Department of Civil Engineering
Ege University, 35100 Bornova, Izmir, Turkey
yalcin.alver@ege.edu.tr

Rolando Armas
Faculty of Engineering
Shinshu University, Japan
rolando.armas@iplab.shinshu-u.ac.jp

Milos Balac
Institute for Transport Planning and Systems (IVT)
ETH Zürich
milos.balac@ivt.baug.ethz.ch

Michael Balmer
Senozon AG
balmer@senozon.com

Mike Batty
Centre for Advanced Spatial Analysis (CASA)
University College London
m.batty@ucl.ac.uk

Gian Ricardo Berkenbrock
Software/Hardware Integration Lab (LISHA)
Universidade Federal de Santa Catarina (UFSC) Joinville
gian.rb@ufsc.br

Davi Guggisberg Bicudo
Universidade Federal de Santa Catarina (UFSC) Joinville
davi.bicudo@me.com

Joschka Bischoff
Transport Systems Planning and Transport Telematics (VSP)
TU Berlin
bischoff@vsp.tu-berlin.de

Frederik Bockemühl
Master's student at Hasselt University
frederik.bockemuhl@student.uhasselt.be

Patrick M. Bösch
Institute for Transport Planning and Systems (IVT)
ETH Zürich
boesch@ivt.baug.ethz.ch

Melanie Bosredon
Centre for Advanced Spatial Analysis (CASA)
University College London
m.bosredon.11@ucl.ac.uk

Paul Bouman
Department of Technology and Operations Management
Rotterdam School of Management (RSM)
Erasmus University Rotterdam
research@pcbouman.nl

Andrew Campbell
CEE Systems and Transportation
University of California, Berkeley
andrew.campbell@berkeley.edu

Artem Chakirov
Future Cities Laboratory
Singapore-ETH Centre
chakirov@ivt.baug.ethz.ch

David Charypar
Institute for Transport Planning and Systems (IVT)
ETH Zürich
dcharypar@gmail.com

Francesco Ciari
Institute for Transport Planning and Systems (IVT)
ETH Zürich
ciari@ivt.baug.ethz.ch

Mario Cools
Local Environment Management & Analysis (LEMA)
University of Liège
mario.cools@ulg.ac.be

Dhirendra Singh
School of Computer Science and I.T.
RMIT University, Melbourne, Australia
dhirendra.singh@rmit.edu.au

Christoph Dobler
Senozon AG
dobler@senozon.com

Thibaut Dubernet
Institute for Transport Planning and Systems (IVT)
ETH Zürich
thibaut.dubernet@ivt.baug.ethz.ch

Alexander Erath
Future Cities Laboratory
Singapore-ETH Centre
erath@ivt.baug.ethz.ch

Sidney Feygin
CEE Systems and Transportation
University of California, Berkeley
sid.feygin@berkeley.edu

Gunnar Flötteröd
Department of Transport Science
KTH Royal Institute of Technology
gunnar.floetteroed@abe.kth.se

Stefan Flügel
Institute of Transport Economics
Norwegian Centre for Transport Research
stefan.flugel@toi.no

Pieter Fourie
Future Cities Laboratory
Singapore-ETH Centre
fourie@ivt.baug.ethz.ch

Daisuke Fukuda
Department of Civil Engineering
Tokyo Institute of Technology
fukuda@plan.cv.titech.ac.jp

Eoghan Furey
National Centre for Geocomputation
NUI Maynooth
eoghan.furey@nuim.ie

Dominik Grether
Transport Systems Planning and Transport Telematics (VSP)
TU Berlin
dominik.grether@alumni.tu-berlin.de

Sashikanth Gurram
Department of Civil & Environmental Engineering
University of South Florida
sgurram@mail.usf.edu

Khandker M. Nurul Habib
Department of Civil Engineering
University of Toronto
khandker.nurulhabib@utoronto.ca

Walter J. Hernández B.
Centro de Computación Gráfica
Universidad Central de Venezuela, Caracas
walter.hernandez@ciens.ucv.ve

Johannes Illenberger
Transport Network Development and Transport Models (GSV)
DB Mobility Logistics AG
johannes.illenberger@deutschebahn.com

Johan W. Joubert
Department of Industrial and Systems Engineering
University of Pretoria
johan.joubert@up.ac.za

Julia Kern
Mathematical Optimization and Scientific Information
Zuse Institute Berlin
kern@zib.de

Benjamin Kickhöfer
Transport Systems Planning and Transport Telematics (VSP)
TU Berlin
kickhoefer@vsp.tu-berlin.de

Hubert Klüpfel
Maleto
hubert@maleto.de

Peter Kucirek
TMG Travel Modelling Group, Toronto
peter.kucirek@alum.utoronto.ca

Gregor Lämmel
Institute for Advanced Simulation (IAS)
Forschungszentrum Jülich GmbH
g.laemmel@fz-juelich.de

Aonghus Lawlor
Insight Centre for Data Analytics
University College Dublin
aonghus.lawlor@insight-centre.org

Seungjae Lee
Departement of Transportation Engineering
University of Seoul
sjlee@uos.ac.kr

Maxime Lenormand
Instituto de Fisica Interdisciplinar y Sistemas Complejos (IFISC)
Campus Universitat de les Illes Balears
maxime@ifisc.uib-csic.es

Milan Lovric
Department of Technology and Operations Management
Rotterdam School of Management (RSM) Erasmus University Rotterdam
lovric.milan@gmail.com

Jiangshan Ma
Shanghai Maritime University
tonny.achilles@gmail.com

Michal Maciejewski
Division of Transport Systems
Poznan University of Technology
michal.maciejewski@put.poznan.pl

Gavin McArdle
National Centre for Geocomputation
Maynooth University
Gavin.McArdle@nuim.ie

Tatyana Mikheeva
Department of Transportation Organization and Management
Samara State Aerospace University, Samara, Russia
mikheevati@its-spc.ru

Sudatta Mohanty
CEE Systems and Transportation
University of California, Berkeley
sudatta.mohanty@berkeley.edu

Daisuke Murakami
Center for Global Environmental Research
National Institute for Environmental Studies, 16-2, Onogawa, Tsukuba, Ibaraki, 305-8506, Japan
murakami.daisuke@nies.go.jp

Mehmet Metin Mutlu
Department of Civil Engineering
Ege University, 35100 Bornova, Izmir, Turkey
mmetinm@gmail.com

Héctor E. Navarro U.
Centro de Computación Gráfica
Universidad Central de Venezuela, Caracas
hector.navarro@ciens.ucv.ve

Andreas Neumann
Senozon Deutschland GmbH
earlier: Transport Systems Planning and Transport Telematics (VSP)
TU Berlin
neumann@senozon.de

Pelin Onelcin
Department of Civil Engineering
Ege University, 35100 Bornova, Izmir, Turkey
pelin.onelcin@ege.edu.tr

Sergio Arturo Ordóñez Medina
Future Cities Laboratory
Singapore-ETH Centre
ordonez@ivt.baug.ethz.ch

Lin Padgham
School of Computer Science and I.T.
RMIT University, Melbourne, Australia
lin.padgham@rmit.edu.au

Miguel Picornell
Nommon Solutions and Technologies
miguel.picornell@nommon.es

Abdul R. Pinjari
Department of Civil & Environmental Engineering
University of South Florida
apinjari@usf.edu

Alexei Pozdnoukhov
CEE Systems and Transportation
University of California, Berkeley
alexeip@berkeley.edu

Marcel Rieser
Senozon AG
rieser@senozon.com

Nadine Rieser-Schüssler
Ernst Basler + Partner AG
earlier: Institute for Transport Planning and Systems (IVT), ETH Zürich
nadine.rieser@ebp.ch

Daniel Röder
Senozon Deutschland GmbH
roeder@senozon.de

Nicole Ronald
Department of Infrastructure Engineering
University of Melbourne
nicole.ronald@unimelb.edu.au

Ismaïl Saadi
Local Environment Management & Analysis (LEMA)
University of Liège
ismail.saadi@ulg.ac.be

Oleg Saprykin
Department of Transportation Organization and Management
Samara State Aerospace University, Samara, Russia
saprykinon@gmail.com

Olga Saprykina
Department of Transportation Organization and Management
Samara State Aerospace University, Samara, Russia
olga_grineva_@mail.ru

Joan Serras
Centre for Advanced Spatial Analysis (CASA)
University College London
j.serras@ucl.ac.uk

Hajime Seya
Graduate School for International Development and Cooperation
Hiroshima University
hseya@hiroshima-u.ac.jp

Chunfu Shao
School of Traffic and Transportation
Beijing Jiaotong University, Beijing, China
cfshao@bjtu.edu.cn

Norihito Shinkai
Regional Futures Research Center Co. Ltd.
shinkai@refrec.jp

David Strippgen
Interactive Systems & Game Technologies
Hochschule für Technik und Wirtschaft (HTW)
david.strippgen@htw-berlin.de

Amy L. Stuart
Department of Civil & Environmental Engineering and Department of Environmental & Occupational Health
University of South Florida
astuart@health.usf.edu

Jacques Teller
Local Environment Management & Analysis (LEMA)
University of Liège
Jacques.Teller@ulg.ac.be

Theresa Thunig
Transport Systems Planning and Transport Telematics (VSP)
TU Berlin
thunig@vsp.tu-berlin.de

Alejandro Tirachini
Transport Engineering Division, Civil Engineering Department
Universidad de Chile
alejandro.tirachini@ing.uchile.cl

Camilo Vargas-Ruiz
Centre for Advanced Spatial Analysis (CASA)
University College London
camilo.ruiz@ucl.ac.uk

Waldemar Walerjanczyk
Division of Transport Systems
Poznan University of Technology
waldemar.walerjanczyk@put.poznan.pl

Rashid A. Waraich
Institute for Transport Planning and Systems (IVT)
ETH Zürich
waraich@ivt.baug.ethz.ch

Adam Weiss
Department of Civil Engineering
University of Toronto
adam.weiss@utoronto.ca

Kaoru Yamada
Oriental Consultants Global Co. Ltd.
yamada-kr@oriconsul.com

Yoshiki Yamagata
Center for Global Environmental Research
National Institute for Environmental Studies, 16-2, Onogawa, Tsukuba, Ibaraki, 305-8506, Japan
yamagata@nies.go.jp

Elvira B. Yaneza
College of Computer Studies
Xavier University-Ateneo de Cagayan de Oro City, Philippines
eyaneza@xu.edu.ph

Mogeng Yin
CEE Systems and Transportation
University of California, Berkeley
mogengyin@berkeley.edu

Vassilis Zachariadis
Centre for Advanced Spatial Analysis (CASA)
University College London
v.zachariadis@ucl.ac.uk

Lun Zhang
Transport Information Engineering
Tongji University Shanghai, China
lun_zhang@tongji.edu.cn

Chengxiang Zhuge
Department of Geography
University of Cambridge
earlier: School of Traffic and Transportation, Beijing Jiaotong University, Beijing, China
cz293@cam.ac.uk

Dominik Ziemke
Transport Systems Planning and Transport Telematics (VSP)
TU Berlin
ziemke@vsp.tu-berlin.de

Michael Zilske
Transport Systems Planning and Transport Telematics (VSP)
TU Berlin
zilske@vsp.tu-berlin.de

Copy-Editing

Karen Ettlin
karen.ettlin@datazug.ch

Introduction

The book is intended to give new MATSim users a quick start in running MATSim. It also provides more experienced MATSim users and MATSim developers with information on how to extend MATSim by plugging in available modules (e.g., the contributions), or by programming against the MATSim API (Application Programming Interface) to implement their own MATSim extensions. Another of this book's goals is to contextualize the methods used in MATSim within a broader theoretical background. By compiling our conceptual insights on MATSim gained over the years, the book also contributes to methodological discussions on joint microsimulation of travel demand and traffic flow, a relatively new field, or, more generally, spatial demand and its congestion generation.

The book is divided into four parts, focused on *using* (Part I), *extending* (Part II), and *understanding* (Part III) MATSim, while simultaneously providing practical, technical, and methodological information. The last part of the book (Part IV) then presents an array of MATSim scenarios that have been created around the world.

Part I: Using MATSim

This part enables users to run MATSim with only the config file, a population and a network. They are given general information to assess whether MATSim is a suitable tool and method for their specific research question.

Chapter 1 introduces the MATSim basics, including its underlying co-evolutionary principle and its traffic flow model. Chapter 2 shows the MATSim novice how to set up and run a basic MATSim scenario. Scoring is central to MATSim; a full chapter, Chapter 3, scrutinizes scoring. Chapter 4 lists the config file options available for basic scenarios containing config file, a population and a network.

Part II: Extending MATSim

This part presents technical information on how to extend the base functionality of MATSim by additional input data beyond config file, population and network, as well as by programming against the API.

Chapter 5 introduces MATSim's modular architecture. It also explains how to use the available modules introduced in Chapters 6 through 42. Chapter 43 describes modules that were important in the past but whose development was discontinued. Chapter 44 briefly describes MATSim organization, i.e., its development process, code structure, the team and the community, and summarizes their development tools. Chapter 45 goes one step further and explains to readers how to write their own MATSim extensions, and how to then contribute them to MATSim, including details about points where MATSim can be extended; it also digs a bit deeper and provides details about the very central MATSim concept of events. Explanations about how to inject alternative or additional modules and how in general to write MATSim scripts in Java is also found here.

Part III: Understanding MATSim

This part presents theoretical aspects underlying the previous two parts. For example, the MATSim score is no longer simply denoted by S without interpretation, but is here contextualized within the discrete choice framework (Chapter 49) and becomes related to utility, commonly denoted by U. The first chapter, Chapter 46 starts with a summary of MATSim's history, written by Kai Nagel and Kay W. Axhausen. Chapter 47 then elaborates on agent-based traffic assignment and qualitatively contextualizes MATSim within classical concepts. Here, the focus is on development from static to dynamic traffic assignment and, finally, agent-based traffic assignment. Chapter 48 quantitatively contextualizes MATSim within classical concepts by presenting it as a fundamentally stochastic tool, based on random distributions and understandable as a Monte Carlo engine. Chapter 50 analyzes MATSim's traffic flow model in relation to kinematic waves, while Chapter 51 provides an economic view on MATSim.

Part IV: Scenarios

At this point, when readers have a complete picture of MATSim and are ready to set up their own real-world MATSim scenario, Chapters 52 through 96 show them the numerous and highly varied scenarios that have been implemented around the world.

The book concludes with a discussion of promising research avenues (Chapter 97).

Related Material

The book concentrates on the more stable aspects of MATSim application and development. In the future, revisions of Chapters 1 to 5 will be presented once a year. Additional material is referenced from `http://matsim.org`, for example under `http://matsim.org/docs`, `http://matsim.org/javadoc`, `http://matsim.org/extensions`, `http://matsim.org/faq`, or `http://matsim.org/issuetracker`.

PART I

Using MATSim

CHAPTER 1

Introducing MATSim

Andreas Horni, Kai Nagel and Kay W. Axhausen

1.1 The Beginnings

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

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

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

How to cite this book chapter:
Horni, A, Nagel, K and Axhausen, K W. 2016. Introducing MATSim. In: Horni, A, Nagel, K and Axhausen, K W. (eds.) *The Multi-Agent Transport Simulation MATSim*, Pp. 3–8. London: Ubiquity Press. DOI: http://dx.doi.org/10.5334/baw.1.

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

1.2 In Brief

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

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

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., 2005b), mode choice (Grether et al., 2009), or destination choice (Horni et al., 2012b) into the iterative loop.

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 52). During iterations, this initial demand is optimized individually by each agent. Every agent possesses a memory containing a fixed number of day plans, 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 51).

In every iteration, prior to the simulation of the network loading with the MATSim *mobsim (mobility simulation)* (e.g., Cetin, 2005), each agent selects a plan from its memory. This selection is dependent on the plan *scores*, which are computed after each mobsim run, based on the executed plans' performances. A certain share of the agents (often 10 %) are allowed to clone the selected plan and modify this clone (*replanning*). For the network loading step, multiple mobsims are available and configurable (see Horni et al., 2011b, and Section 4.3 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.,

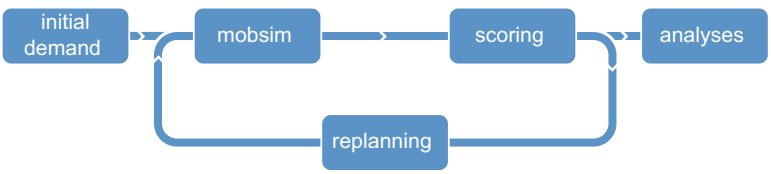


Figure 1.1: MATSim loop, sometimes called the MATSim cycle.

2005b), 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 3.

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 convergence criteria appropriate for deterministic algorithms; for a discussion of possible approaches for the MATSim situation, see Sections 47.3.2.2 and 48.2 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 5 and following.

MATSim is strongly based on events stemming from the mobsim. Every action in the simulation generates an event, which is recorded for analysis. These event records can be aggregated to evaluate any measure at the desired resolution. The event architecture is detailed in Section 45.2.5.

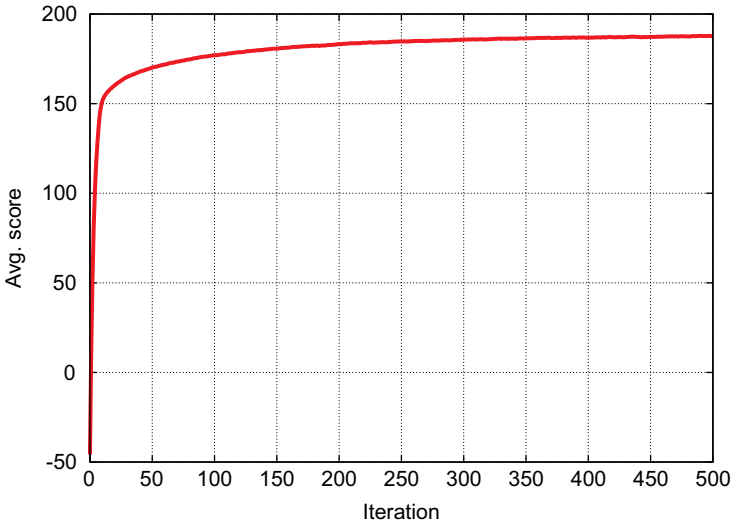


Figure 1.2: Typical score progress.

1.3 MATSim's Traffic Flow Model

MATSim provides two internal mobsims: QSim and JDEQSim (Java Discrete Event Queue Simulation); in addition, external mobility simulations can be plugged in. Some years ago, the DEQSim 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 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 have been added (Chapter 12), backward-moving gaps, as investigated by Charypar (2008), are included in JDEQSim, but only available on an *experimental basis* for QSim (Section 97.5). Interactions between different modes are described in Section 4.6 and Chapter 21.

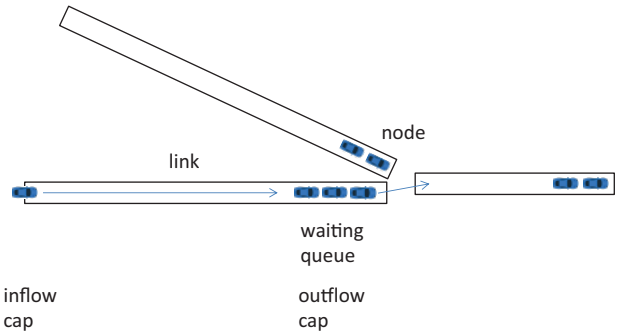


Figure 1.3: Traffic flow model.

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.

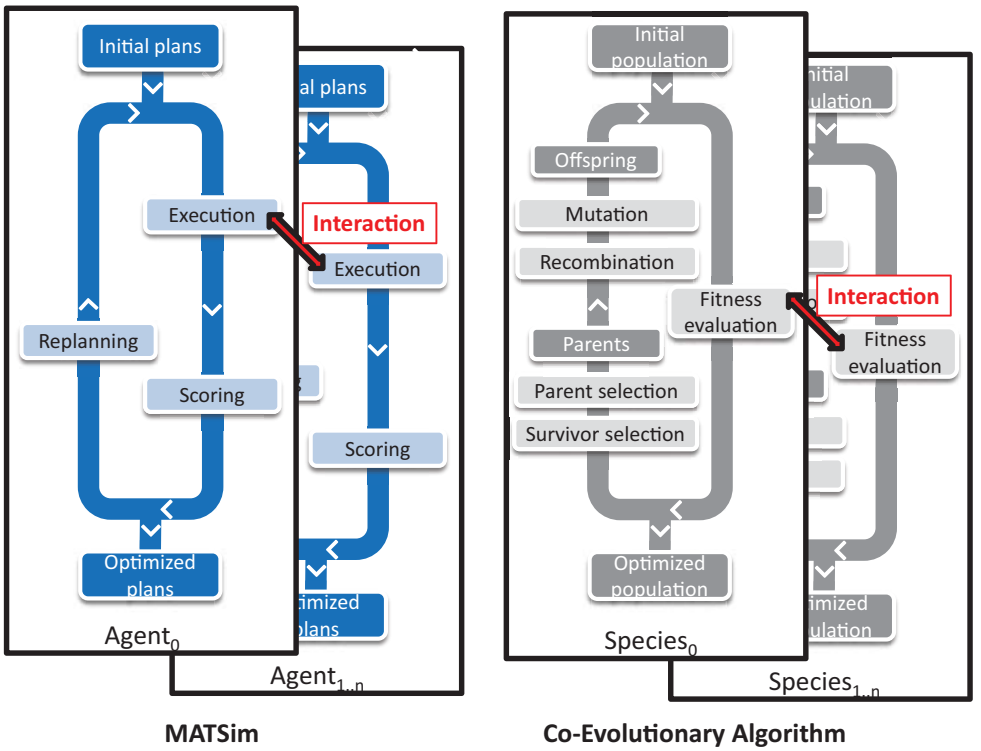


Figure 1.4: The co-evolutionary algorithm in MATSim.

CHAPTER 2

Let's Get Started

Marcel Rieser, Andreas Horni and Kai Nagel

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

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

2.1 Running MATSim

2.1.1 Setting Up MATSim

To run MATSim, you must install the Java SE (Java Standard Edition) that complies with the appropriate MATSim version. At this time, this is Java SE 7.

Download of the release You also 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 "release" link under `http://matsim.org/downloads`. Unzip results in the **MATSim directory tree**. Continue with Section 2.1.2.

The MATSim directory tree on the web If you want to look at the development version, or look at things without downloading and installing a zip file: On GitHub, the root of the **MATSim directory tree** (i.e., excluding so-called contribs and playgrounds) is at `https://github.com/matsim-org/matsim/tree/master/matsim`.

How to cite this book chapter:
Rieser, M, Horni, A and Nagel, K. 2016. Let's Get Started. In: Horni, A, Nagel, K and Axhausen, K W. (eds.) *The Multi-Agent Transport Simulation MATSim*, Pp. 9–22. London: Ubiquity Press. DOI: http://dx.doi.org/10.5334/baw.2.

Download of nightly builds If you prefer to use the more up-to-date, but less stable, *nightly builds,* you should download, via the same URL (Uniform Resource Locator) `http://matsim.org/downloads`,

- the MATSim JAR (Java ARchive) file (usually tagged with the revision number `MATSim_ryyyy.jar`), and
- the required external libraries (`MATSim_libs.zip`). Unzipping this collection of 3rd-party libraries, you should then get a directory `libs`, with several JAR files inside. If the directory `libs` is in the same directory as the MATSim JAR file, the libraries are found automatically and do not have to be added manually to the `classpath`.

Maven A relatively new feature is that one can use MATSim as an Apache Maven plugin; both release versions and snapshots are available. See again `http://matsim.org/downloads` for more information. For someone who has used Apache Maven before, this is probably the best option. In this case, one may use the simple Java programming approach of Section 5.1.1.4 to get started.

2.1.2 Running MATSim

When this book was written, only the nightly built MATSim JAR file could be started by double-clicking. A minimal GUI (Graphical User Interface), as shown in Figure 2.1, opens and the MATSim run can be configured and started. This feature will appear in the releases, starting with version 0.8.

For the release 0.7, MATSim does not provide a GUI; thus, you must be able to handle and access a command line tool. In Linux or Mac OS X, this is typically a Terminal application; in Microsoft Windows, the Power Shell or Command Prompt. At the command prompt type the following command in one line, but substitute the correct paths:

On Linux or Mac OS X, something like:

```
java -Xmx512m -cp /path/to/matsim.jar org.matsim.run.Controler /path/
   to/config.xml
```

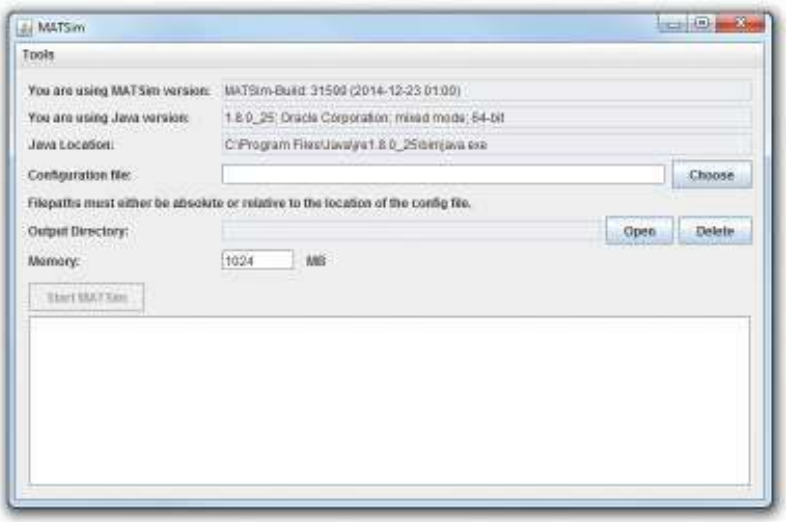


Figure 2.1: Minimal MATSim GUI.

On Windows, an example command could be:

```
java -Xmx512m -cp C:\MATSim\matsim.jar org.matsim.run.Controler
   C:\MATSim\input\config.xml
```

Such a command consists of multiple parts:

- `java` tells the system that you want to run Java.
- `-Xmx512m` tells Java that it should use up to 512 MB (Megabyte) of memory. This is typically enough to run the small examples. For larger scenarios, you might need more memory, e.g., `-Xmx3g` would allow Java to use up to 3 GB (Gigabyte) of RAM (Random Access Memory).
- `-cp /path/to/matsim.jar` tells Java where to find the MATSim code.
- `org.matsim.run.Controler` specifies which class (think of an "entry point") should be run. In most cases, the default MATSim `Controler` is the class you will need to run simulations.
- `/path/to/config.xml` tells MATSim which config file is to be used.

2.1.3 Configuring MATSim

MATSim is configured in the config file, building the connection between the user and MATSim and containing a settings list that influences how the simulation behaves.

All configuration parameters are simple pairs of a parameter `name` and a parameter `value`. The parameters are grouped into logical groups; one group has settings related to the `Controler`, like the number of iterations, or another group has settings for the mobsim, e.g., end time of the mobsim. As shown in Chapter 5, numerous MATSim modules can be added to MATSim and configured by specifying the respective configuration file section.

The list of available parameters and valid parameter values may vary from release to release. Although we try to keep this stable, software changes, mainly new features, may cause settings to change. For a list of all available settings available with the version you are working with, run the following command:

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

This command will create a new config file `fullConfig.xml`, containing all available parameters, along with their default values and often an explanatory comment, making it easy 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. See `http://matsim.org/javadoc` → main distribution → `CreateFullConfig` for more information.

A fairly minimal config file contains the following information:

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

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

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

<module name="planCalcScore" >
   <parameterset type="activityParams" >
```

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

For a working example, see the MATSim directory tree (cf. 2.1.1) under `examples/tutorial/config/example1-config.xml`.

In the example, supply is provided by the network and demand by the plans file. Typical input data is described in Section 2.2.2. The specification that the first and last iteration are the same, means that no replanning of the demand is performed. What *is* executed is the mobsim (Figure 1.1), followed by each executed plan's performance scoring. To function, the scoring needs to know, from the config file, all activity types used in the plans and the typical duration for each activity type.

Further configuration possibilities are described in Chapter 4.

2.2 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. It presents a minimal example scenario and briefly explains units, conventions and coordinate systems used in MATSim. Then, hints on practical data requirements are provided.

2.2.1 Units, Conventions, and Coordinate Systems

2.2.1.1 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 the coordinate system uses, allowing MATSim to calculate beeline distances. As the much used UTM (Universal Transverse Mercator) projected coordinate systems (see Section 2.2.1.3) use meters as the unit of distance, this is the most commonly used distance unit in MATSim.

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

Money Money is unit-free. Units are implicitly given by the marginal utility of money (cf. Equation (3.4) below). Thus, when one moves from Germany to Switzerland, the parameter β_c must be changed from "utility per Euro" to "utility per Swiss Franc".

2.2.1.2 Conventions

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

2.2.1.3 Coordinate Systems

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

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

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

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

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

This allows MATSim to work with your coordinates and convert them whenever needed.

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

As an alternative, MATSim can also parse the description of a coordinate system in the WKT (Well-Known Text) format.

2.2.2 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 day plans.

Thus, `population.xml` and `network.xml` might get quite large. To save 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 acknowledges whether files are compressed, or should be written compressed, based on file name.

2.2.2.1 An Outlook on Extending MATSim in Part II of this Book

Chapter 7 provides some information about MATSim's technical tools for initial input generation. With the basic setting, MATSim agents perform their activities on a specific link. If further information about activity locations needs to be specified, this can be carried out with facilities described in Section 6.4. Further, for the *simulation* of public transport, the base scenario must be extended by additional files as shown in Section 16.4.1 and Chapter 16. Count data are a common evaluation measure in transport planning. In MATSim, count data can be provided for the simulation, as shown in Section 6.3.

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

2.2.2.2 `network.xml`

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

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

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

Each element has an identifier `id`. Nodes are described by an `x` and a `y` coordinate value (also see Sections 2.2.1.3 and 7.1). Links have more features; the `from` and `to` attributes reference nodes and describe network geometry. Additional attributes describe traffic-related link aspects:

- The `length` of the link, typically in meters (see Section 2.2.1).
- 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 alternating `to` and `from` attributes (see links with id 2 and 3 in the listing above).

2.2.2.3 `population.xml`

File Format MATSim travel demand is described by the agents' day 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 day 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, check the `examples/equil` directory in the MATSim directory tree (cf. Section 2.1.1).

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

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

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

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

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

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

The following items can be used for simplification:

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

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

2.2.3 Typical Output Data

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

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

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

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

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

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

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

The following output files are created for specific iterations:

Events: Every action in the simulation is recorded as a MATSim event, be it an activity start or change of network link; see Fig. 2.2. Each event possesses one or multiple attributes. By default, the time when the event occurred is included. Additionally, information like the ID of the agent triggering the event, or the link ID where the event occurred, could be included. The events file is an important base for post-analyses, like the visualizers. Events are discussed in detail in Section 45.2.5.

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.

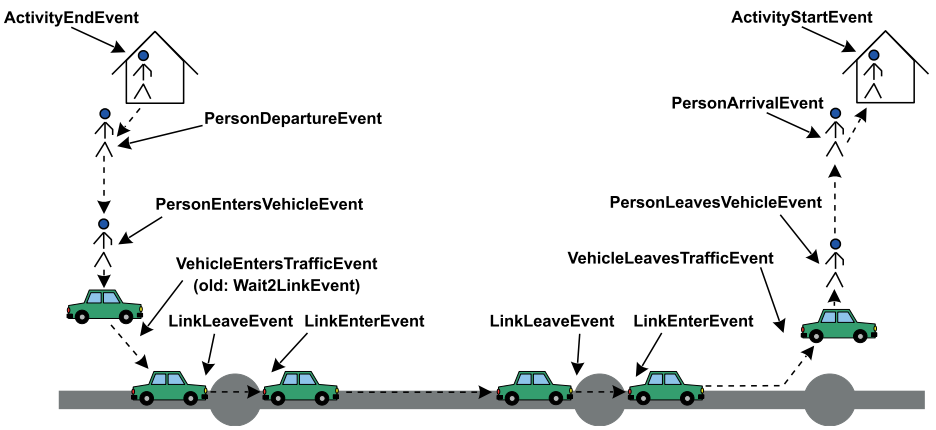


Figure 2.2: Mobsim events.

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

2.2.4 An Example Scenario

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

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

The scenario network is shown in Figure 2.3.

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

"strategy" section of the config file As shown in the config file excerpt below, this scenario uses replanning. 10 % of the agents reroute their current route (module `ReRoute`). The remaining 90 % select their highest score plan for re-execution in the current iteration (module `BestScore`). Plans are deleted from the agent's memory if it is full, defined by `maxAgentPlanMemorySize`. By default, the plan with the lowest score is removed; this is configurable and currently being researched (see Section 97.3).

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

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

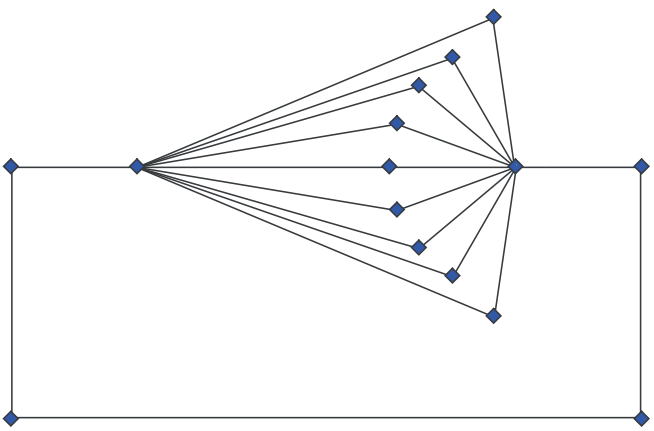


Figure 2.3: Equil scenario network.

```
    <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 3. 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 2.2.2.3) must be defined in the `planCalcScore` section of the config file.

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

"controler" section of the config file The scenario is run for 10 iterations, writes the output files to `./output/equil` (Section 2.2.3) and uses QSim as the mobsim (more on mobsims in Section 1.3, 4.3 and 11).

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

Visualization Simulation results can be visualized with Via (Chapter 33) or OTFVis (On The Fly Visualizer) (Chapter 34).

2.2.5 Data Requirements

2.2.5.1 Population and Activity Schedules

Demand estimation is an important component of MATSim. That means that, in theory, only demand components that do *not* change from one simulated average working day to the next need to be provided to MATSim. Examples are: population and its residential and working locations. In practice, however, MATSim is not yet prepared to endogenously model complete travel demand. Sequence and preferred durations of activities, for example, must be provided as input. As a result, all travel demand choices not covered by the MATSim loop have to be exogenously estimated.

For population generation, two possibilities exist: the comfortable way is to translate a full population census and the slightly 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 based on e.g., Swiss Federal Statistical Office (BFS) (2000) and Müller (2011a).

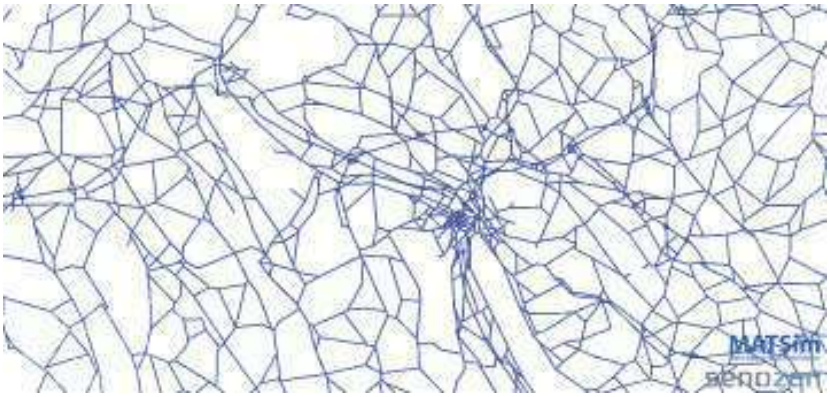
Travel demand is usually derived from surveys: for Switzerland, from the microcensus (Swiss Federal Statistical Office (BFS), 2006). Newer data sources, such as GPS or smartphone travel diaries, are currently being investigated (e.g., Zilske and Nagel, 2015).

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.

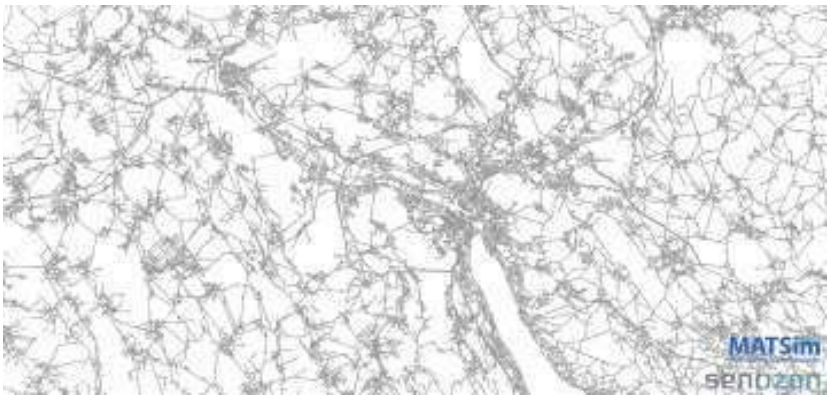
Having generated the residential population of the study area, additional demand components might be necessary, for example, cross-border and freight traffic. As these components often cannot be endogenously modeled, MATSim offers the feature to handle different subpopulations differently (Section 4.5). 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.

2.2.5.2 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 as initial explorative simulation runs, while the latter are often used for policy runs, usually offering far more details, such as bike and even pedestrian links. Data are available from official sources like federal offices, free sources, such as OSM (OpenStreetMap), and commercial sources, including navigation network providers.



(a) Planning network.



(b) Navigation network..

Figure 2.4: Zürich networks

2.2.6 Example Scenario Input Data

Some example scenarios are included in the MATSim main distribution, in the directory "examples".

More pre-packaged scenarios can be found under `http://www.matsim.org/datasets`.

2.3 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. *Always start with and test a small example.*
2. *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 4.3, this requires adaptation of parameters, in particular, the mobsim's `flowCapacityFactor` and `storageCapacityFactor` factors. As shown in Part II, Section 6.3, sample scenarios also require parameter adaption for count data comparisons.
3. *If your set-up does not work any more,* immediately *go back to a working version and proceed from there in small steps.*
4. *Check* `logfileWarningErrors.log`.
5. *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`.
6. *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.
7. *Search for documentation via* `http://matsim.org/javadoc`.
8. *Search for the latest tutorial via* `http://matsim.org/docs`.

CHAPTER 3

A Closer Look at Scoring

Kai Nagel, Benjamin Kickhöfer, Andreas Horni and David Charypar

3.1 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 47 and 49. 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, ingeneral,

How to cite this book chapter:

Nagel, K, Kickhöfer, B, Horni, A and Charypar, D. 2016. A Closer Look at Scoring. In: Horni, A, Nagel, K and Axhausen, K W. (eds.) *The Multi-Agent Transport Simulation MATSim*, Pp. 23–34. London: Ubiquity Press. DOI: http://dx.doi.org/10.5334/baw.3.

select good plans. Here, "good" means "able to obtain a high score in the mobsim/scoring". Fortunately, due to its evolutionary concept, the approach is fairly robust: the innovation step does not always have to generate good solutions; it is sufficient if *some* of the solutions are good and lead to a high score.

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

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

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

However, because MATSim is based on complete day plans, the application of choice models for parts of day plans only (for example, mode choice) is not straightforward, as detailed in Section 97.4.4. 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 3.2). This scoring function was, at times, modified, extended, or replaced for specific investigations (Section 3.5). Readily applicable estimates for a full-day utility function are not yet available, as discussed in Section 97.4.4.

3.2 The Current Charypar-Nagel Utility Function

3.2.1 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. It is, however, almost certainly not appropriate for activity dropping and activity addition (see Section 3.3).

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

$$S_{plan} = \sum_{q=0}^{N-1} S_{act,q} + \sum_{q=0}^{N-1} S_{trav,mode(q)} \tag{3.1}$$

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.

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

$$S_{act,q} = S_{dur,q} + S_{wait,q} + S_{late.ar,q} + S_{early.dp,q} + S_{short.dur,q} \,. \tag{3.2}$$

The individual contributions are defined as follows:

- The expression

$$S_{dur,q} = \beta_{dur} \cdot t_{typ,q} \cdot \ln(t_{dur,q}/t_{0,q}) \tag{3.3}$$

  is the utility of performing activity q, where opening times of activity locations are taken into account. $t_{dur,q}$ is the performed activity duration, β_{dur} is related to the marginal utility of activity duration (or marginal utility of time as a resource, the same for all activities; see Section 3.2.4), and $t_{0,q}$ is the duration when utility starts to be positive.
- The expression

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

  denotes waiting time spent, for example, in front of a still-closed store; β_{wait} is the so-called *direct* (see Section 3.2.4) marginal utility of time spent waiting; and $t_{wait,q}$ is the waiting time. We recommend leaving β_{wait} at zero; also see Section 3.2.5.
- The expression

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

  specifies the late arrival penalty, where $t_{start,q}$ is the activity starting time 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).
- The expression

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

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

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

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

The config syntax (config version v2) is approximately

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

Travel Travel disutility for a leg q is given as

$$S_{trav,q} = \begin{array}{c} C_{mode(q)} + \beta_{trav,mode(q)} \cdot t_{trav,q} + \beta_m \cdot \Delta m_q \\ +(\beta_{d,mode(q)} + \beta_m \cdot \gamma_{d,mode(q)}) \cdot d_{trav,q} + \beta_{transfer} \cdot x_{transfer,q} \end{array} \tag{3.4}$$

where:

- $C_{mode(q)}$ is a mode-specific constant.
- $\beta_{trav,mode(q)}$ is the *direct* (see Section 3.2.4) marginal utility of time spent traveling by mode. Since MATSim uses and scores 24-hour episodes, this is in addition to the marginal utility of time as a resource (again, see Section 3.2.4).
- $t_{trav,q}$ is the travel time between activity locations q and $q+1$.
- β_m is the marginal utility of money (normally positive).
- Δm_q is the change in monetary budget caused by fares, or tolls for the complete leg (normally negative or zero).
- $\beta_{d,mode(q)}$ is the marginal utility of distance (normally negative or zero).
- $\gamma_{d,mode(q)}$ is the mode-specific monetary distance rate (normally negative or zero).
- $d_{trav,q}$ is the distance traveled between activity locations q and $q+1$.
- $\beta_{transfer}$ are public transport transfer penalties (normally negative).
- $x_{transfer,q}$ is a 0/1 variable signaling whether a transfer occurred between the previous and current leg.

The config syntax (config version v2) is approximately

```
<module name="planCalcScore" >
   <param name="marginalUtilityOfMoney" value="1.0" />
   <param name="utilityOfLineSwitch" value="-1.0" />
   <parameterset type="modeParams" >
      <param name="mode" value="car" />
      <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 (3.4) is the direct utility contribution of travel; see Section 3.2.4 for the the full indirect utility as well as the relation to the VTTS (Value of Travel Time Savings), and Chapter 51 for a more general discussion.

Note that 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.

3.2.2 *Illustration*

Figure 3.1 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 all other things (particularly travel times) are equal, 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 51.2), 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. Clearly, this holds only when all other things remain constant, particularly travel times.

3.2.3 *The "Wrapping Around" of the Utility Function*

The MATSim mobsim typically starts at midnight and runs until all plans have reached their final activity. By itself, the mobsim, is not limited to a day. However, as already stated in Section 3.2.1, 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.

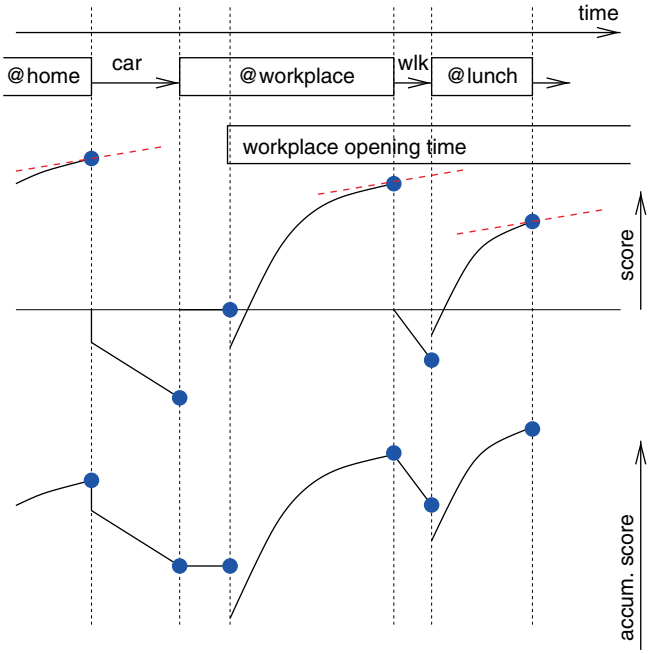


Figure 3.1: Illustration of the scoring function. TOP: Individual contributions of activities and legs. BOTTOM: Score accumulation over a day.

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

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

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

$$mUTTS = -\frac{\partial}{\partial t_{trav}} S_{trav} + \frac{\partial}{\partial t_{dur}} S_{dur}.$$

which is

$$mUTTS = -\beta_{trav} + \beta_{dur} \cdot \frac{t_{typ,q}}{t_{dur,q}} \tag{3.5}$$

and at the typical duration of an activity

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

where it can be imagined q is the activity immediately following the trip (cf. Section 51.2). The marginal utility of travel time savings, $mUTTS$, can thus be defined as the indirect effect on the overall time budget, corrected by an offset β_{trav} that denotes how much better, or worse, it is to spend that time traveling, rather than "doing nothing".[1] To differentiate β_{trav} from the indirect effect, it is sometimes called **direct marginal utility** of time spent traveling.

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

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

and at the typical duration of an activity

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

This is important for calibration of the utility function.

3.2.5 The Resulting Modeling of Schedule Delay Costs

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

[1] 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 51.2.

by $mUWTS = -\beta_{wait} + \beta_{dur} \cdot \frac{t_{typ,q}}{t_{dur,q}}$: Even when the direct marginal utility of waiting, β_{wait}, equals zero, then "doing nothing" still eats into the overall time budget and thus incurs the same opportunity cost of time as traveling does. Intuitively, one can imagine that one must leave the previous activity earlier to have a longer waiting time, thus reducing the score of the previous activity.

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

Arriving Late Arriving late incurs a marginal utility of β_{late}, typically negative. Here, no additional opportunity cost of time is involved. Intuitively, arriving later implies having left the previous activity later. That is: the current activity is shortened by the same amount that the previous activity was extended, leaving the overall score unaffected (cf. Section 51.2).

Vickrey Parameters As a result, the Vickrey parameters of α (marginal penalty for arriving early), β (marginal penalty for traveling) and γ (marginal penalty for arriving late) (as defined by Arnott et al., 1990) are consistent with the following equations:

$$\begin{aligned} -\beta_{wait} + \beta_{dur} \cdot \frac{t_{typ,q}}{t_{dur,q}} &= \alpha \\ -\beta_{trav} + \beta_{dur} \cdot \frac{t_{typ,q}}{t_{dur,q}} &= \beta \\ -\beta_{late} &= \gamma\,. \end{aligned} \tag{3.6}$$

3.3 Implementation Details

This section summarizes 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.

3.3.1 Zero Utility Duration

The duration when an activity's utility is exactly zero is computed by the somewhat cryptic expression

$$t_{0,q} := t_{typ,q} \cdot \exp\left(-\frac{10\,h}{t_{typ,q} \cdot prio}\right), \tag{3.7}$$

where *prio* is a configurable parameter. This is designed so that all activities with the same value of *prio* obtain, at their typical duration, i.e., when $t_{dur,q} = t_{typ,q}$, the same utility value of $10 \cdot \beta_{dur}$, with the idea that this makes them equally likely to be dropped in a time shortage situation (Charypar and Nagel, 2005).[2] However, this does not work as intended, since activities receiving this utility value from 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

[2] Starting from Equation (3.3) and inserting Equation (3.7), one obtains

$$\begin{aligned} S_{dur,q}\Big|_{t_{dur,q}=t_{typ,q}} &= \beta_{dur} \cdot t_{typ,q} \cdot \ln\left(\frac{t_{typ,q}}{t_{typ,q} \cdot \exp\left(-10\,h/(t_{typ,q} \cdot prio)\right)}\right) \\ &= \beta_{dur} \cdot t_{typ,q} \cdot \ln\left(\exp\left(10h/(t_{typ,q} \cdot prio)\right)\right) = 10\,h \cdot \beta_{dur}/prio\,, \end{aligned}$$

which is indeed the same for all activities with the same value of *prio*.

first, which is clearly not plausible. See Section 97.4 for a discussion of alternatives. In the meantime, 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.

3.3.2 Negative Durations

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 3.2.3). In this situation, it can happen that an agent arrives at the last activity of the plan at a later 24-hour-time than when the first activity ended. For example, an agent could stay at home until 3 am (end of first activity), then go through her daily plan including a very late party, and return home at 6 am the next morning (Figure 3.2). In this case, the duration of the wrap-around home activity would be *minus* three hours. 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 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. The current approach is to take the slope of the expression $\beta_{dur} \cdot t_{typ,q} \cdot \ln(t_{dur,q}/t_{0,q})$ when it crosses zero, and extend this towards minus infinity (Figure 3.3).

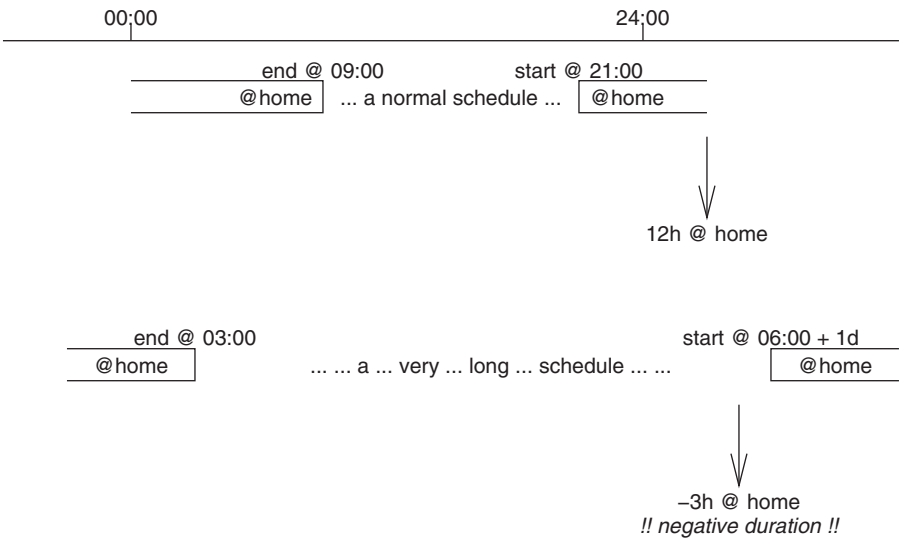


Figure 3.2: Illustration of wrap-around scoring. TOP: Normal situation. BOTTOM: Situation where final activity starts at a later time of day then when the first activity ended, resulting in negative duration.

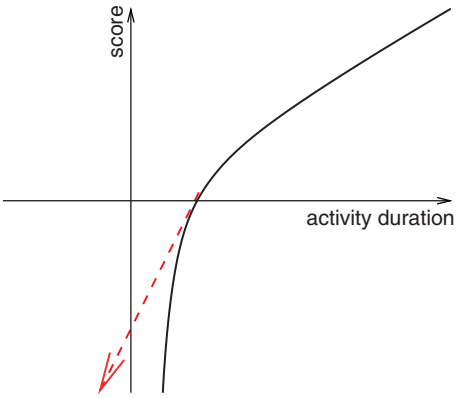


Figure 3.3: Extending the slope when the utility function crosses the zero line to negative durations.

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.

3.3.3 Score Averaging

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

$$S^k = \alpha\, S + (1-\alpha)\, S^{k-1}\,, \tag{3.8}$$

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

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

in the config file.

Non-executed plans just keep their score.

3.3.4 Forcing Scores to Converge

For many situations, both practical and theoretical (see Section 47.3.2.2), it is desirable that each plan's score converges to its expectation value. Equation (3.8) will not achieve that; it just dampens the fluctuations. A well-known approach to force convergence to the expectation value is MSA (Method of Successive Averages):

$$S^m = \frac{1}{m}\, S + \frac{m-1}{m}\, S^{m-1}. \tag{3.9}$$

This resembles Equation (3.8), with two important differences: (1) The fixed blending parameter α is now replaced by a variable $1/m$, and (2) 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="..." />
```

This is plausibly used together with innovation switch off (Section 4.5.3), meaning that MSA operates on a fixed set of plans.

3.4 Typical Scoring Function Parameters and their Calibration

The current MATSim default values are

$$\begin{array}{lrl} \beta_m & = & 1 \quad utils/monetaryunit \\ \beta_{dur} & = & 6 \quad utils/h \\ \beta_{trav,mode(q)} & = & -6 \quad utils/h \\ \beta_{wait} & = & 0 \quad utils/h \\ \beta_{short.dur} & = & 0 \quad utils/h \\ \beta_{late.ar} & = & -18 \quad utils/h \\ \beta_{early.dp} & = & 0 \quad utils/h. \end{array} \tag{3.10}$$

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

An additional insight is that, in many of the systems that we model, traveling does not seem to be less convenient than "doing nothing". Thus, the *direct* marginal utility of traveling, β_{trav}, is 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:[3]

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

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

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

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

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

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

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

   It is important to understand this: Even if this value is set to zero, traveling by car will be implicitly punished by the opportunity cost of time: If you are traveling by car, you cannot perform an activity; thus, you are (marginally and approximately) losing β_{dur}. See Section 3.2.4.

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

   For example, if your logit model says something like

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

   then

   $$\beta_{dur} = 6\,, \quad \beta_{tt,car} = 0\,, \text{ and } \beta_{tt,pt} = -1\,.$$

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

[3] Different groups have different systems; this one is typical for VSP, although it uses ideas from Michael Balmer.

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

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

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

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

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

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

3.5 Applications and Extensions

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 52.

Special utility functions have been developed for car sharing (see Chapter 22), social contacts and joint trips (see Chapter 28), parking (see Chapter 13), road pricing (see Chapter 15) and destination innovation (see Chapter 27), 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 97.4.4), inclusion of agent-specific preferences (Section 97.4.5) and application of alternative utility function forms (Section 97.4).

CHAPTER 4

More About Configuring MATSim

Andreas Horni and Kai Nagel

This chapter describes configuration options that can be used together with the three basic elements: config file, population and network. Part II discusses various options to extend MATSim beyond these three elements, sometimes using only additional files, or using additional JAR files beyond the MATSim core JAR file, by writing "scripts in Java" or by adding or replacing functionality.

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 2.3, these files come with comments explaining configuration options. This is often the best source for configuration options.

4.1 MATSim Data Containers

4.1.1 Network

The config file section `network` specifies which network file will be used in the simulation (Section 2.1.3 and 2.2.2.2). Further configuration options, e.g., specification of time-variant networks, are presented in Section 6.1.

4.1.2 Population

The config file section `plans` specifies which population file with its day plans will be used (Section 2.1.3 and 2.2.2.3). Further configuration options, e.g., specification of arbitrary agent attributes or subpopulations, are presented in Section 6.2.

Further MATSim containers are described in Chapter 6.

How to cite this book chapter:

Horni, A and Nagel, K. 2016. More About Configuring MATSim. In: Horni, A, Nagel, K and Axhausen, K W. (eds.) *The Multi-Agent Transport Simulation MATSim*, Pp. 35–44. London: Ubiquity Press. DOI: http://dx.doi.org/10.5334/baw.4.

4.2 Global Modules and Global Aspects

4.2.1 Controler

The controler is an indispensable module for running MATSim; its parameters are set in the `controler` config file section. 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 4.3). The routing algorithm is defined here by using

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

Possibilities for extending the `Controler` functionality are given in Chapter 45.

4.2.2 Events

Events are continuously generated, reporting on all activities in the mobsim, as discussed in more detail in Section 45.2.5.

Please note that, besides these mobsim events, there is a less prominent type of events, namely `ControlerEvents`, which are created by the `Controler` to report on its current state. `ControlerEvents` are also further explained in Section 45.2.5.

4.2.3 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 Setting The `global` section contains

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

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

Parallel Event Handling The config file section `parallelEventHandling` 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' eventually 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 older hardware (e.g., Apple Macbook Pro from 2010), using all three remaining cores—in addition to the parallel event handling—led to negligible performance gains but left the machine useless for interactive tasks such as normal office work. For new 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). No experiences with new servers are currently available.
- HPCC (High-Performance Computing Clusters) are often available to researchers, allowing access to high-quality machines with reduced management overhead. 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 thus 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 throughput. In addition, for both calibration and "what-if" scenario exploration, one typically needs to run a large number of simulations with different parameters or input data. As total RAM memory is usually not an issue on a cluster, it is often more efficient to run a large number of simulations simultaneously with a low number of threads, rather than a low number of simulations with lots of threads.

4.2.4 Global

In the config file section `global`, the simulation's random seed, the "global" number of Java threads (see Section 4.2.3) and the coordinate system (cf. Section 2.2.1) 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 file, 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.

4.3 Mobility Simulations

An overview of MATSim mobility simulations is given by Dobler and Axhausen (2011).

4.3.1 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 `qsim` config file section. Important parameters are: By specifying

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

QSim can be run in parallel, see Section 4.2.3. Importantly, the `qsim` 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.

Currently, QSim is implemented as a single-queue model (see Chapter 50). Back-propagating gaps as discussed in Section 1.3 are available experimentally (see Section 97.5) and configurable with the parameter

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

As shown in Section 4.6.1, QSim can handle multimodal scenarios.

A somewhat ancient configuration parameter is the stuck time. It 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 was 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.

4.3.2 JDEQSim

JDEQSim (Waraich et al., 2009) was used for project *KTI Frequencies* (Balmer et al., 2010). It is is a Java reimplementation of DEQSim (Waraich et al., 2009; Charypar et al., 2007b, 2009) and provides parallel event handling, but no parallel simulation (Balmer et al., 2010, p.11). Back-propagating gaps (Section 1.3) are supported, but traffic lights, public transport and within-day replanning are not.

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

4.4 Scoring

The config file section `planCalcScore` specifies the parameters used for scoring agents' plans (Section 2.1.3); parameters are explained in Chapter 3.

4.5 Replanning Strategies

Replanning strategies are the basic innovation modules available in MATSim. We do not call them *choice* modules, although they are involved in people's choice making. The choice process is performed over the iterations with an *implicit* choice set and is not based on explicit probability function drawing. One can differentiate between modules that affect the set of plans that each agent holds, and others that only select between these plans. For a detailed discussion of MATSim in choice modeling context, see Chapter 49.

All strategy modules are called by configuring the strategy module in the configuration file as shown in the following example.

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

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

By specifying the parameter `subpopulation`, replanning strategies can be applied to distinct subpopulations: e.g.,

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

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

Please note that combining strategy modules that are extensions (see Section 5.1.1), like destination innovation together with public transport, may not always work as expected. Combine them with care and contact the mailing list if you are unsure.

4.5.1 Plans Generation and Removal (Choice Set Generation)

4.5.1.1 Time Innovation

Time innovation is applied by defining its parameters in the config file section `TimeAllocationMutator` and by adding

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

plus its weight to the strategy modules.

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

4.5.1.2 Route Innovation

Route innovation is applied by adding

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

plus its weight to the strategy modules, and by specifying the routing algorithm in the `controler` config file section (Section 4.2.1). MATSim routing is described by Lefebvre and Balmer (2007).

4.5.1.3 Mode Innovation

Mode innovation is applied by adding[1]

```
<param name="strategyName"
   value="{ChangeLegMode | ChangeSingleLegMode |
   SubtourModeChoice}" />
```

plus its weight to the strategy modules. In the config file, a section with one of the mode innovation strategies needs to be added, i.e.,

```
<module name="{changeLegMode | changeSingleLegMode |
subtourModeChoice}" >
   ...
</module>
```

`ChangeLegMode` 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` module is required. Optionally, car availability is respected. `ChangeSingleLegMode` randomly picks one of a person's plans and changes one single leg's (picked randomly) mode of transport. In contrast to `ChangeLegMode`, it allows for multiple modes in one plan. By default, supported modes are: driving a car and using public transport. Also, this module can (optionally) respect car availability.

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

4.5.1.4 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 yet well investigated, cf. Section 97.3. Per default, the plan with the lowest score is removed if the agent's memory is full.

4.5.2 Plan Selection (Choice)

Selectors and their weight are also added to the strategy modules

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

[1] The names may be changed into `ChangeTripMode` and `ChangeSingleTripMode`, please keep your eyes open.

Selectors work as follows:

- `KeepLastSelected` keeps the plan selected in the previous iteration.
- `BestScore` selects the plan with the highest score from the previous iteration.
- `SelectExpBeta` performs MNL (Multinomial Logit Model) selection between plans. It can be configured by the `BrainExpBeta` parameter from the scoring config group[2] being the scale parameter in discrete choice models, as shown in Equation 49.2. We recommend keeping this parameter at its default value of 1.0.
- `ChangeExpBeta` changes to a different plan, with probability dependent on $e^{\Delta_{score}}$, where Δ_{score} is the score difference between the two plans. This will also sample from an MNL (see Sec. 47.3.2.1).
- `SelectRandom` performs random selection between the plans.
- `SelectPathSizeLogit` selects an existing plan according to the path size logit described by Frejinger and Bierlaire (2007). It can be configured by the `PathSizeLogitBeta` parameter from the scoring config group.[3] This selector has never been investigated systematically.

Note that the `BestScore` should be used with care; it tends to get stuck with sub-optimal plans. Plans badly rated 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 `SelectRandom`.

4.5.3 Innovation Switch-Off

For theoretical (Section 47.3.2.3) reasons, it makes sense to eventually switch off the innovative modules, thus keeping the set of plans for each agent fixed from then on. This behavior can be configured by

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

It makes sense to use this together with MSA averaging of the scores (Section 3.3.4).

4.6 Other Modes than Car

The MATSim software began with the car mode of transport, since it was then the main mode in many regions. The idea of integrating other modes has always been a theme.

The following sections describe current MATSim multi-modal capabilities. The material covers not only options that can be enabled with just config options, but also gives an overview of multi-modal extensions, described in Part II of the book.

4.6.1 QSim Side

4.6.1.1 Multiple Vehicular Modes on the Same Network

The approach described so far fails as soon as more than one vehicle type is involved. Therefore, recently the ability to allow multiple modes on the same network was introduced. It is defined by the `qsim` config option of type

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

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

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

It is currently not possible to generate different vehicle types from the config alone; one either needs to provide a vehicles file (see Section 6.6 and Section 11.1), or write a script-in-Java to generate the vehicle fleet (again see Section 11.1).

4.6.1.2 So-Called Teleportation

All modes *not* registered with the QSim as "main modes" will be teleported. That is, the QSim will, without problems, process legs such as

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

The QSim will generate a departure event (for events, see Section 2.2.3) 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 3), it can also be left out of the route (the situation in most set-ups).

4.6.1.3 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; for example, the traveler 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, 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; 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. In these cases, their interactions are captured by the simulation.

4.6.1.4 Departure Handlers

It is possible to register a separate departure handler for each mode; see Section 45.2.8 for the syntax. There are also pre-configured extensions using this approach:

- The "multimodal" contribution 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 21).
- The public transport extension moves all registered modes with specific public transit vehicles (see Chapter 16).
- The dynamic transport systems contribution will eventually be able to move a `taxicab` mode with taxis (see Chapter 23).

4.6.2 Routing Side

The previous Section 4.6.1 has described how the QSim handles various modes when they are requested by the plans. Correspondingly, it now needs to be considered how non-car plans, or more specifically non-car routes inside non-car legs, are generated.

4.6.2.1 Network Modes

The following syntax defines modes for which the router should generate network routes, i.e., routes that contain a sequence of links to follow:

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

The above configuration specifies that plans containing

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

as well as

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

will be treated by the network router.

As of the writing of this text, the router will route all these modes on the "car" links of the network. This means that, say, denominating some links as "car only" or "truck only" will not be picked up by the current router.[4]

Note that, per the network file DTD (Document Type Description), "car" is the default mode of each link as long as long as the link's mode field is not explicitly filled.

4.6.2.2 Teleportation ...

... 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).

... with Teleported Mode Speed Setting, in the above, something like

```
   <param name="teleportedModeFreespeedFactor" value="null" />
   <param name="teleportedModeSpeed" value="4.167" />
   <param name="beelineDistanceFactor" value="1.3" />
```

will, instead, 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

[4] Check `https://matsim.atlassian.net/browse/MATSIM-330` for developments.

to car travel times. One disadvantage: this approach does not take obstacles like water or mountain areas, into account for the teleported modes.

4.6.2.3 Other Routing Options

It is possible to register separate routers for specific modes. This syntax is discussed in Section 45.2.7. The pre-configured extensions and contributions discussed in Section 4.6.1.4, "multimodal", public transport, taxis, come with corresponding routers.

In addition, the so-called "matrix based pt router" (Chapter 20) 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 algorithms for arbitrary modes, as long as they generate (expected) travel times and (expected) travel distances. As said earlier, the teleportation facility of the QSim will just use these two attributes at face value. Although with such an approach neither congestion nor en-route replanning are or can be included, it is flexible and allows a fully modular addition of arbitrary modes without having to interact with the QSim.

4.6.3 Scoring Side

For all modes mentioned in the plans, a corresponding scoring section must exist. See Section 3.2.1 for an example.

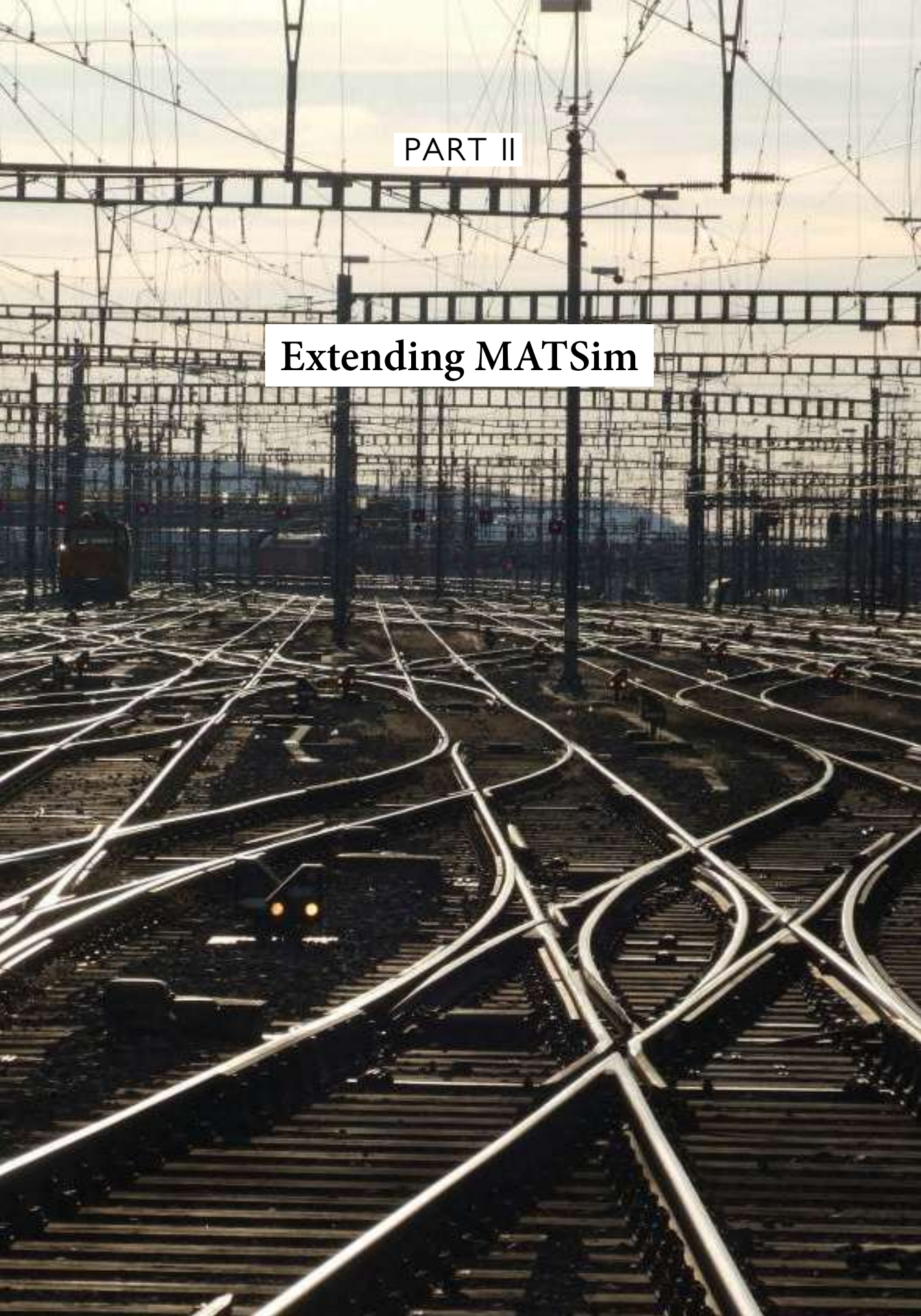
4.7 Observational Modules

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 need to be aggregated to 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 6.3.



PART II

Extending MATSim

CHAPTER 5

Available Functionality and How to Use It

Andreas Horni and Kai Nagel

In this chapter you will learn about possibilities to extend and customize MATSim (Multi-Agent Transport Simulation) through provided functionality. In Chapter 45, you will see how you can hook your own extensions into MATSim.

5.1 MATSim Modularity

MATSim follows a modular concept, but a "module" is not a very specific term;[1] thus, modules can exist at many levels in a software framework. Also in MATSim, a range of different functionality types, such as config functions, replanning components, contributions, or even external tools,[2] are sometimes described as modules. Metaphorically speaking, a module can thus be seen as the greatest common divisor (gcd) of different functionality provided in MATSim. Much more important is understanding the different levels of access stemming from the generally modular architecture.

5.1.1 Levels of Access

MATSim currently provides five levels of access:

1. using the MATSim core only,
2. using the MATSim main distribution,

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

[2] Standalone tools referencing MATSim as a library, such as the network editor, or the visualizer Via.

How to cite this book chapter:
Horni, A and Nagel, K. 2016. Available Functionality and How to Use It. In: Horni, A, Nagel, K and Axhausen, K W. (eds.) *The Multi-Agent Transport Simulation MATSim*, Pp. 47–52. London: Ubiquity Press. DOI: http://dx.doi.org/10.5334/baw.5.

3. using MATSim main distribution, contributions and possibly extensions,
4. writing "scripts in Java" and finally
5. writing your own extensions.

5.1.1.1 Using the Core 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[3] 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.

5.1.1.2 Using MATSim Main Distribution

The extensions in the MATSim main distribution are, by design, very close to the MATSim core, thus requiring even less configuration than for contributions, as shown below. Often, providing additional files together with a respective config file entry is sufficient to use them; required steps are described below, case by case. Extensions contained in the main distribution are listed in a separate section at `http://matsim.org/extensions`.

5.1.1.3 Using One or More Contribs or Other Extensions

Contributions are in a separate part of the repository, separate from the MATSim main distribution. The documentation is not yet fully organized; information about contributions and other extensions can be found at `http://matsim.org/extensions`. For the contributions, there are also release versions and nightly builds, which can be found by following the links at `http://matsim.org/downloads`.

In general, contributions should provide main methods for use. We may eventually provide clickable jar files here as well, but for the time being, contributions need to be bundled with core MATSim (and potentially other contributions). As shown at `http://www.matsim.org/docs/extensions`, the syntax is roughly

```
java -Xmx2000m -cp MATSim.jar:contrib/contrib.jar org.matsim.contrib.run.RunXxx
   config.xml
```

where

- `-Xmx2000m` increases the Java heap space, so that most MATSim runs fit in,
- `MATSim.jar` needs to be replaced by a relative or absolute path to the MATSim jar to be used,
- `contrib/contrib.jar` needs to be replaced by a relative or absolute path to the contribution jar to be used,

[3] This has worked since winter 2014/15 and should be in the 0.8.x release.

- `org.matsim.contrib.run.RunXxx` needs to be replaced by the full Java class name containing the desired main method (given by the contribution documentation), and
- `config.xml` needs to be replaced by a relative or absolute path to a config file, which may contain additional sections specific to the contribution.

It is possible to combine several contributions in this way, provided someone has made a corresponding main method available. This can, in principle, be done relatively quickly, so those wishing to run studies with combinations of existing contributions, but without programming skills, can ask someone with those skills and with access to the repository for help.

5.1.1.4 *Writing "Scripts in Java"*

The contributions are written so that they can be plugged into MATSim via extension points (see Chapter 45). If a specific combination or configuration of modules is not (yet) available, one can write it. The syntax is roughly:

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

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

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

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

Extension points, especially at `(*)` and `(**)`, are described in more detail in Chapter 45.

5.1.1.5 *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, when writing an extension, one should use the extension points described in Chapter 45, since this is the only way an extension can later become a contribution.

5.1.2 *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,[4] VISUM (Verkehr In Städten – UMlegung),[5] EMME (Equilibre Multimodal Multimodal Equilibrium), or SUMO (Simulation of Urban Mobility) (via the TraCI interface)[6] for plug-ins. Scala (SCAlable LAnguage) was discussed for MATSim, but ultimately, it was decided to just use Java itself as the scripting language, with the advantage that users between development and MATSim application do not need to learn two languages. In addition, the TU (Technische Universität) Berlin team can continue to teach Java both as an entry point to MATSim and as a general professional skill.

5.2 An Overview of Existing MATSim Functionality

Figure 5.1 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

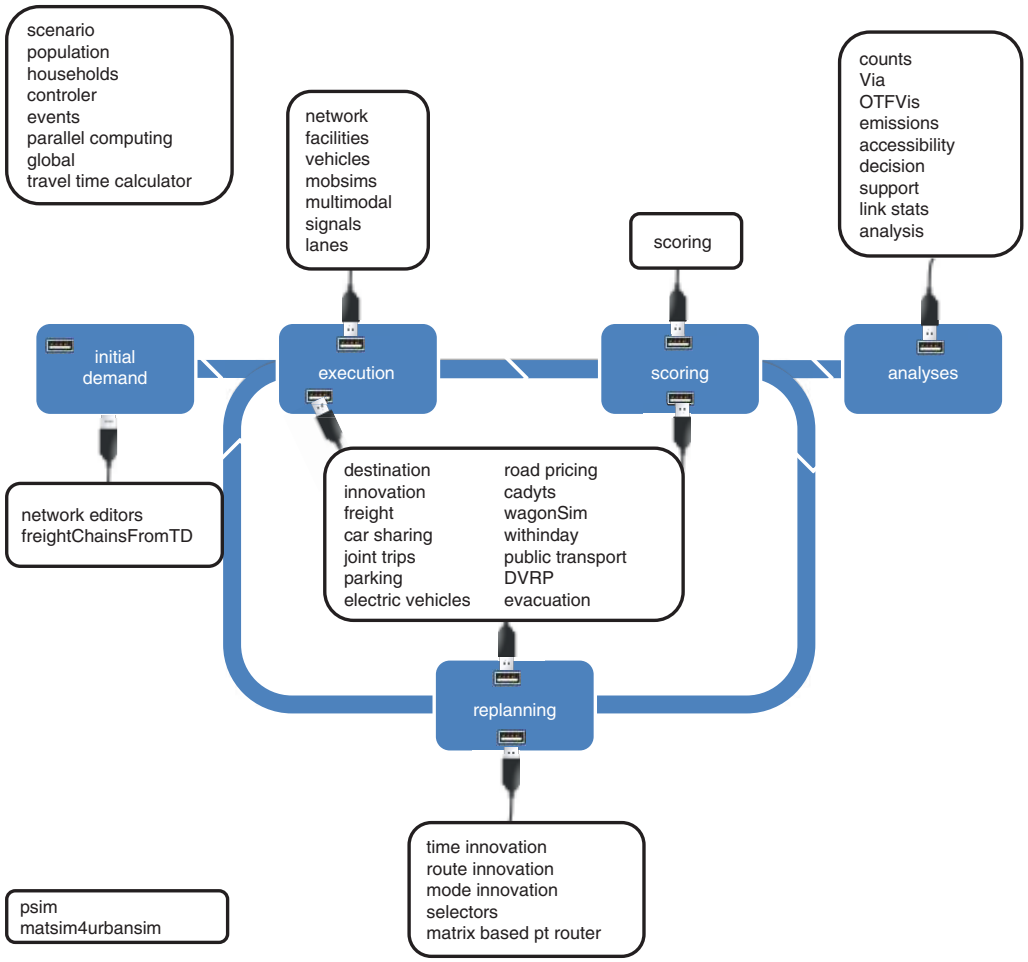


Figure 5.1: MATSim functionality.

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

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).

The technical details for module usage, in particular, the parameter sets, are described at `http://matsim.org`, especially `http://matsim.org/javadoc` and `http://matsim.org/extensions`.

As a result of the distributed and project- and dissertation-driven MATSim contribution process (see Chapter 44), modules are often implemented for a specific practical purpose, leading to limitations of the respective module. For example, modules might only work for a specific mode, or for a defined calling order. Normally, additional effort is needed to generalize the module; in consequence, the combination of a specific module with other functionality is often not a straightforward 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 5.1.

Global Modules and Global Aspects	Section 4.2
Controler	Section 4.2.1
Events	Section 4.2.2
Parallel Computing	Section 4.2.3
Global	Section 4.2.4
MATSim Data Containers	Section 4.1 and Chapter 6
Network	Section 4.1.1 and 6.1
Population	Section 4.1.2 and 6.2
Counts	Section 6.3
Facilities	Section 6.4
Households	Section 6.5
Vehicles	Section 6.6
Scenario	Section 6.7
Network Editors	
MATSim JOSM Network Editor	Chapter 8
Map-to-Map Matching Editors in Singapore	Chapter 9
The "Network Editor" Contribution	Chapter 10
Observational Modules	Section 4.7
Travel Time Calculator	Section 4.7.1
Link Stats	Section 4.7.2
Scoring	Section 4.4
Basic Strategy Modules	Section 4.5
Time Innovation	Section 4.5.1.1
Route Innovation	Section 4.5.1.2
Mode Innovation	Section 4.5.1.3
Selectors	Section 4.5.2
Mobsims	
QSim	Section 4.3.1 and Chapter 11
JDEQSim	Section 4.3.2
Individual Car Traffic	
Signals and Lanes	Chapter 12
Parking	Chapter 13

Continued on next page

Electric Vehicles	Chapter 14
Roadpricing	Chapter 15
Other Modes Besides Individual Car	
Public Transport	Chapter 16
The "Minibus" Contribution	Chapter 17
Semi-Automatic Tool for Bus Route Map Matching	Chapter 18
Events-Based Public Transport Router	Chapter 19
matrix-based pt router	Chapter 20
Multi-Modal Contribution	Chapter 21
Car Sharing	Chapter 22
Dynamic Transport Systems	Chapter 23
Commercial Traffic	
Freight Traffic	Chapter 24
wagonSim	Chapter 25
freightChainsFromTravelDiaries	Chapter 26
Additional Choice Dimensions	
Destination Innovation	Chapter 27
Joint Trips and Social Networks	Chapter 28
Socnetgen	Chapter 29
Within-Day Replanning	
Within-day Replanning	Chapter 30
Belief Desire Intention (BDI) Framework	Chapter 31
Automatic Calibration	
Cadyts	Chapter 32
Visualizers	
Via Visualizer	Chapter 33
OTFVis Visualizer	Chapter 34
Analysis	
Accessibility	Chapter 35
Emissions	Chapter 36
Interactive Analysis and Decision Support	Chapter 37
The "analysis" contrib	Chapter 38
Computational Performance Improvements	
PSim	Chapter 39
Other Modules	
Evacuation	Chapter 41
MATSim4UrbanSim	Chapter 42

Table 5.1: MATSim functionality overview.

SUBPART ONE

Input Data Preparation

CHAPTER 6

MATSim Data Containers

Marcel Rieser, Kai Nagel and Andreas Horni

6.1 Time-Dependent Network

The network container was already described in Section 4.1.1. 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 the `examples/equil-extended` directory in the MATSim directory tree.

Alternatively, network change events can be added directly to the code. An example can be found in the `RunTimeDependentNetworkExample` class under `http://matsim.org/javadoc` → main distribution.

How to cite this book chapter:
Rieser, M, Nagel, K and Horni, A. 2016. MATSim Data Containers. In: Horni, A, Nagel, K and Axhausen, K W. (eds.) *The Multi-Agent Transport Simulation MATSim*, Pp. 55–60. London: Ubiquity Press. DOI: http://dx.doi.org/10.5334/baw.6. License: CC-BY 4.0

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

6.2 Person Attributes and Subpopulations

The population container was also already discussed earlier, in Section 4.1.2. A powerful extension of a standard population can be achieved by specifying further agent attributes in an `ObjectAttributes` file input to MATSim by the parameter `inputPersonAttributesFile`.

See `http://matsim.org/javadoc` → main distribution → `RunSubpopulationsExample` class for an example. That example looks as if coding in Java is necessary, but this is really not the case; Java is just used to generate the subpopulations, which could also be done by other means.

6.3 Counts

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

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

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

```
<?xml version="1.0" encoding="UTF-8"?>
<counts name="example" desc="example counting stations" year="2015">
   <count loc_id="2" cs_id="005">
      <volume h="1" val="10.0"></volume>
      <volume h="2" val="1.0"></volume>
      <volume h="3" val="2.0"></volume>
      <volume h="4" val="3.0"></volume>
      <volume h="5" val="4.0"></volume>
      <volume h="6" val="5.0"></volume>
      <volume h="7" val="6.0"></volume>
      <volume h="8" val="7.0"></volume>
      <volume h="9" val="8.0"></volume>
      <volume h="10" val="9.0"></volume>
      <volume h="11" val="10.0"></volume>
      <volume h="12" val="11.0"></volume>
      <volume h="13" val="12.0"></volume>
      <volume h="14" val="13.0"></volume>
      <volume h="15" val="14.0"></volume>
      <volume h="16" val="15.0"></volume>
      <volume h="17" val="16.0"></volume>
      <volume h="18" val="17.0"></volume>
      <volume h="19" val="18.0"></volume>
      <volume h="20" val="19.0"></volume>
      <volume h="21" val="20.0"></volume>
      <volume h="22" val="21.0"></volume>
      <volume h="23" val="22.0"></volume>
      <volume h="24" val="23.0"></volume>
   </count>
</counts>
```

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

It starts with a header containing general descriptive information about the counts, including a year to describe how current the data is. Next, for each link having real world counts data, hourly

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

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

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

6.4 Facilities

Facilities are an optional element of MATSim; some modules, such as the destination innovation module (Chapter 27), depend on it. If MATSim facilities are used, agents perform their activities in a specific facility attached to a network link.

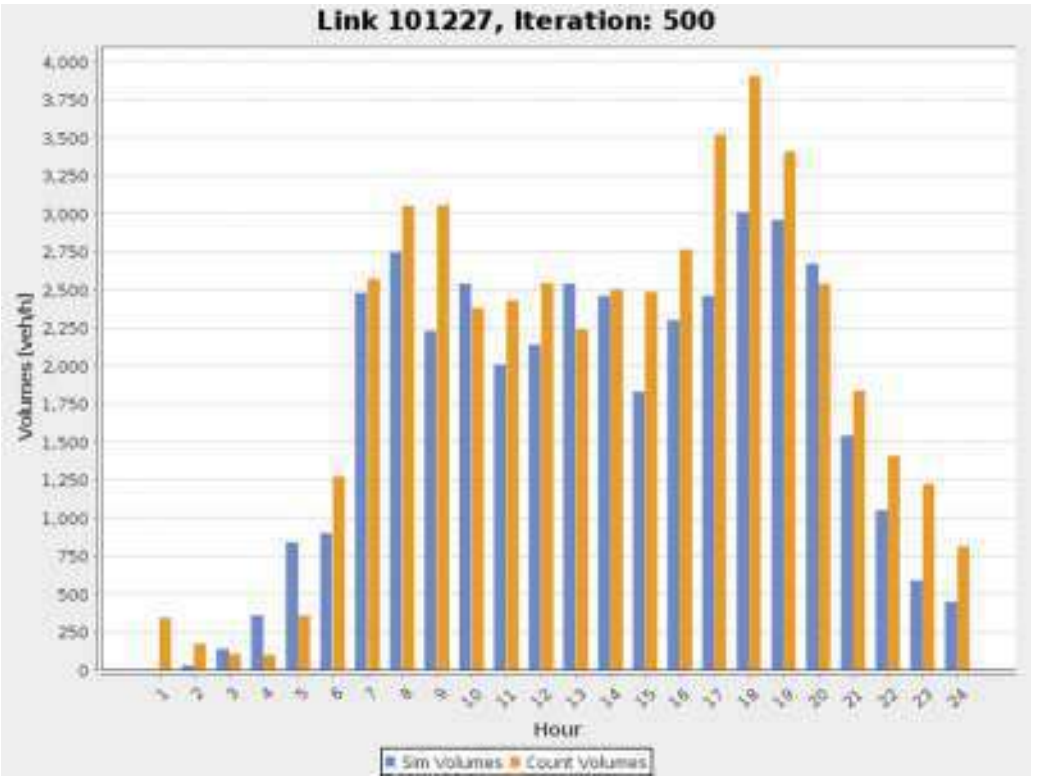


Figure 6.1: Example for a link volumes comparison between simulation and road count values.

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

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

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

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

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

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

6.5 Households

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

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

6.6 Vehicles

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

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

needs to be added.[2]

[1] There used to be an additional "useHouseholds" config switch. In release 0.8.x, that switch will be gone.
[2] There used to be an additional "useVehicles" config switch. In release 0.8.x, that switch will be gone.

Once more, just loading the vehicles does not use them; that needs to be configured separately. See Section 11.1 for details.

6.7 Scenario

`Scenario` is a super-container containing all the other data containers, accessible, for example, as `scenario.getNetwork()`. It used to have configuration options, but these are all gone now, so `Scenario` is only visible once you are programming in Java.

CHAPTER 7

Generation of the Initial MATSim Input

Marcel Rieser, Kai Nagel and Andreas Horni

As explained in Section 2.2, the minimal MATSim input, besides the configuration, consists of the network and population with initial plans. For illustrative scenarios, all three can be generated with a text editor. For more complicated and/or realistic scenarios, they need to be generated by other methods. People with knowledge in a scripting language may use that scripting language to generate the necessary XML files, possibly honoring the MATSim DTDs. 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 5.1.1.4), 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.

How to cite this book chapter:
Rieser, M, Nagel, K and Horni, A. 2016. Generation of the Initial MATSim Input. In: Horni, A, Nagel, K and Axhausen, K W. (eds.) *The Multi-Agent Transport Simulation MATSim*, Pp. 61–64. London: Ubiquity Press. DOI: http://dx.doi.org/10.5334/baw.7.

7.1 Coordinate Transformations in Java

Section 2.2.1.3 has given information about coordinate systems. When programming in Java and MATSim for input data generation, coordinate transformations derived from geotools (Geotools, accessed 2015) can be used. For example,

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

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

7.2 Network Generation

7.2.1 From OpenStreetMap

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

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

   ```
   java -cp osmosis.jar --rb file=xxx.osm.pbf \
      --bounding-box top=47.701 left=8.346 bottom=47.146 right=9.019 \
      completeWays=true --used-node --wb 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

   ```
   java -cp osmosis.jar --rb file=xxx.osm.pbf --tf accept-ways \
      highway=motorway,motorway_link,trunk,trunk_link,primary,primary_link \
      --used-node --wb bigroads.osm.pbf
   ```

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

   ```
   java -cp osmosis.jar --rb file=bigroads.osm.pbf --rb allroads.osm.pbf \
      --merge --wx merged-network.osm
   ```

An example script of how to convert the resulting `merged-network.osm` file into a MATSim network file can be found under `http://matsim.org/javadoc` → main distribution → `RunPNetworkGenerator` class.

7.2.2 From Other Sources

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

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

7.3 Initial Demand Generation

7.3.1 Simple Initial Demand

A simple script to generate a population with a single synthetic person with one initial plan can be found under `http://matsim.org/javadoc` → main distribution → `RunP0nePersonPopulationGenerator`. A somewhat larger synthetic population is generated by `RunPPopulationGenerator`.

Note that coordinates in the population need to be consistent with coordinates in the network. Roughly speaking, coordinates mentioned in the population file need to be in the same range as coordinates mentioned in the network. Note that, in the examples presented here, coordinates of the network generated in Section 7.2.1 are *not* consistent with the demand generated by the `RunP*`-scripts; these need to be adapted accordingly.

7.3.2 Realistic Initial Demand

A script to illustrate the generation of a more realistic population and initial demand can be found under `http://matsim.org/javadoc` → main distribution → `RunZPopulationGenerator`, generating a sample population from a census file and writing it to a file.

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

CHAPTER 8

MATSim JOSM Network Editor

Andreas Neumann and Michael Zilske

8.1 Basic Information

Entry point to documentation:

`http://matsim.org/extensions` → josm-plugin

Invoking the module:

Can be loaded as a plug-in from the JOSM editor.

Selected publications:

Kühnel (2014) (in German)

8.2 Introduction

A plugin for the JOSM (Java Open Street Map Editor) (JOSM, 2014), is available, simplifying the process of creating and editing MATSim networks. This plugin fully integrates with JOSM, benefiting from its built-in functionality.

8.2.1 Features

The MATSim JOSM network editor lets a reader preview, edit and save a MATSim network directly from the map. Basic support for converting and editing public transport networks is implemented. The plug-in allows automatic post-processing of a network by removing unnecessary intermediate nodes and links.

Convert MATSim networks from OSM. Load map data for a selected area directly from the Internet or load it from a local OSM file. Specify conversion parameters and save a MATSim network.

How to cite this book chapter:

Neumann, A and Zilske, M. 2016. MATSim JOSM Network Editor. In: Horni, A, Nagel, K and Axhausen, K W. (eds.) *The Multi-Agent Transport Simulation MATSim*, Pp. 65–66. London: Ubiquity Press. DOI: http://dx.doi.org/10.5334/baw.8.

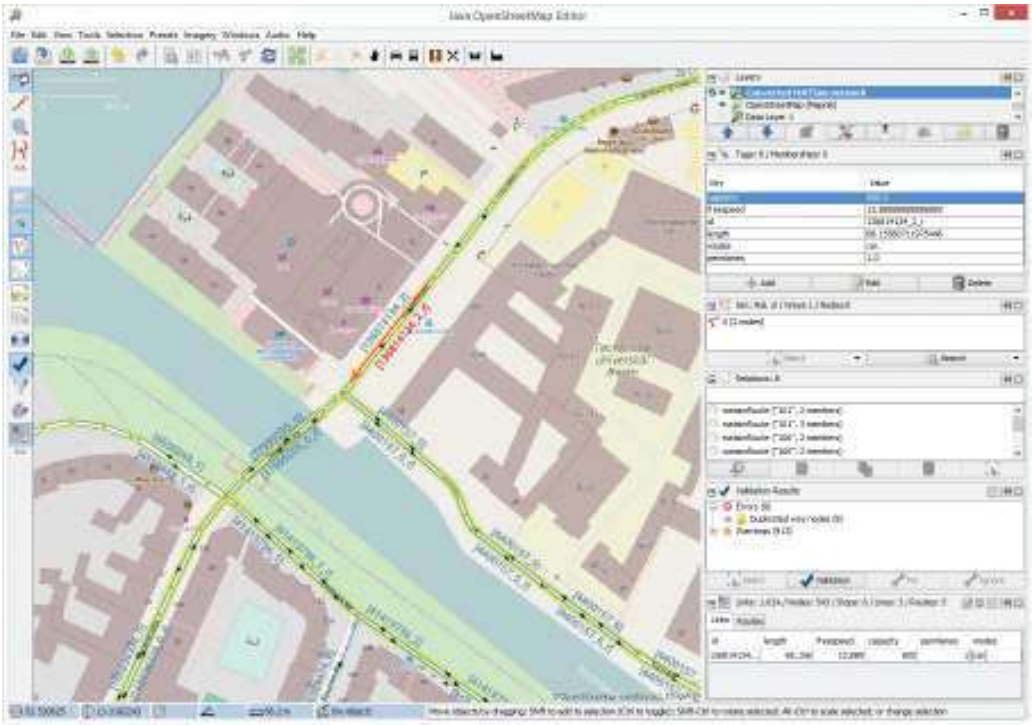


Figure 8.1: JOSM with converted MATSim network and OSM background imagery. Map data taken from OpenStreetMap (2014).

Visualize an existing or newly converted MATSim network along with other data like satellite imagery or other JOSM-supported layers.

Edit an existing or newly converted MATSim network with the available JOSM tools you know. Use the build-in undo and search functions of JOSM. Changes to the underlying OSM data are immediately reflected by the converted MATSim network. Use MATSim-specific presets to minimize errors.

Validate an existing or newly converted MATSim network to comply with requirements of the MATSim network file description. Visualize errors and fix them (automatically).

The next version will support public transport networks.

8.2.2 Installing the Plug-In

You do not need to download the source; it is in the JOSM plug-in repository. Just start JOSM and look for the MATSim plug-in under `Edit...Preferences...Plugins`. Download the list of available plug-ins and search for "matsim". Tick the box, press `ok` and restart JOSM.

8.2.3 Getting the Code

The source code is hosted on github (`https://github.com/matsim-org/josm-matsim-plugin`). Unlike MATSim, the build is not based on Apache Maven, but on Gradle. Editing the `Manifest`, downloading JOSM for compilation and building a flat JAR are easier in Gradle. Use your favorite IDE (Integrated Development Environment) to import the Gradle project and/or see the comments in `build.gradle` for details. You can run JOSM and the plug-in in the debugger.

CHAPTER 9

Map-to-Map Matching Editors in Singapore

Sergio Arturo Ordóñez

9.1 Basic Information

Entry point to documentation:

`http://matsim.org/extensions` → networkEditorSingapore

Invoking the module:

See `http://matsim.org/extensions` → networkEditorSingapore for more information

Selected publications:

Ordóñez Medina (2011a)

For the Singapore scenario and supply data, a high resolution network was obtained from the NAVTEQ company. This network consists of a graph representing every road in the island: very convenient for a high resolution model like MATSim. However, the information on travel capacities and network link free speeds is not accurate. To offset, local authorities provided the network model used for planning, which includes only major roads and simplified intersections, but capacities and free speed are accurately estimated. Figure 9.1 shows lower travel capacities of many primary roads in the navigation model (right), than in the planning model (left).

This section describes a semi-automatic tool developed to match these two network models (Ordóñez Medina, 2011a), allowing updating of navigation network (high-res network) main links/capacities and free speeds with those of the planning network (low-res network).

How to cite this book chapter:

Ordóñez, S A. 2016. Map-to-Map Matching Editors in Singapore. In: Horni, A, Nagel, K and Axhausen, K W. (eds.) *The Multi-Agent Transport Simulation MATSim*, Pp. 67–72. London: Ubiquity Press. DOI: http://dx.doi.org/10.5334/baw.9.

Figure 9.1: Difference in the travel capacities between the Singapore planning network model (left) and a navigation network model (right).

9.1.1 General Procedure

Although many authors try to solve matching problems for two networks in a formal way, this work follows a semi-automatic approach. This means that automatic algorithms will be used to try and solve the problem, but the user knows the solution will not be perfect; some manual work must be done. Hence, interactive tools are also provided to manually improve solutions.

The map-to-map procedure is based on the algorithm developed by Balmer et al. (2005a). It consists of the following steps:

1. Classify nodes according to their topology (e.g., source, sink, one way start, crossing) in both networks.
2. Reduce networks according to previous classification, and save relations to the original nodes.
3. Find crossings (set of close nodes) in both networks and relate them.
4. **Assuming not all crossings were found in the previous step, use the interactive tool shown in the Figure 9.2 to find all crossings in both networks and relate them.**
5. Recognize links or sequences of links joining crossings found in (3) and (4).
6. **Assuming not all links or paths found in the previous step are correct, use the link-link matching interactive tool shown in the Figure 9.3, to find or modify links or sequences of links joining the crossings**
7. Update capacities and free speeds of matched links found in (5) and (6).

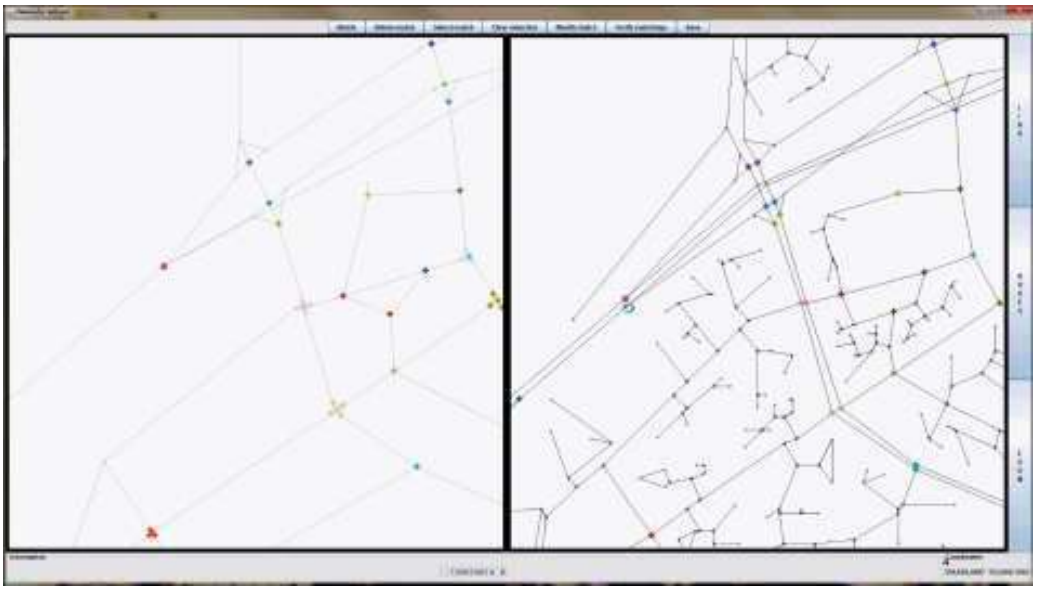


Figure 9.2: Crossing-crossing matching application. A second node, matching the pink node on the (left) low-res network, is selected from the high-res network on the right.

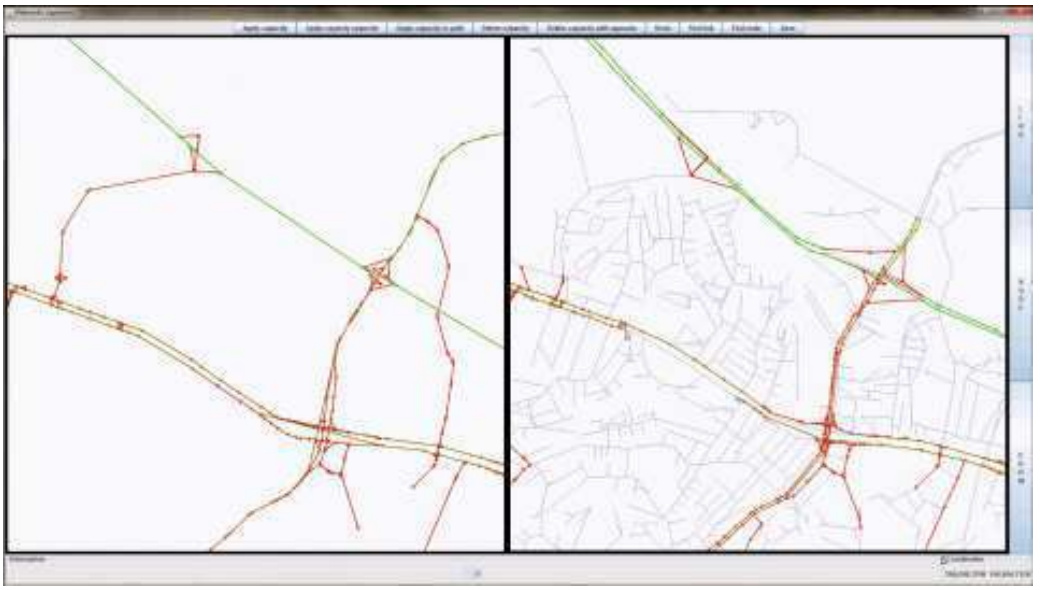


Figure 9.3: Link-link matching application. A shortest path algorithm to select a sequence of right-hand network links will be executed when clicking the destination node.

9.1.2 Interactive Tools Characteristics

As shown in Figure 9.2, the application allows interactive modifying of crossing-crossing relationships. A very similar interactive tool was also developed to modify link-link relationships between the high and low resolution networks. They can be found at the package `playground.sergioo.networksMatcher2012`, in the playgrounds project of MATSim. To run the crossings-crossing application graphic interface, use the class `gui.DoubleNetworkMatchingWindow`, and use the class `gui.DoubleNetworkCapacitiesWindow` for the link-link application. These applications write simple text files of the relationships located. The program found at the class `ApplyCapacities` overwrites capacities and/or free speeds, according to simple text files and writes the new resulting network XML file. This multiple-steps design enables running interactive applications several times, or in parallel. The interactive tools' developed functional requirements and quality attributes are:

- Visualization: Two navigation networks are displayed in two modes. The first mode splits the window in two, showing each network on one side and maintaining them at the same geographical position and zoom when navigating. The second superimposes both networks in the same window, with only one active. Selected elements are drawn in different colors. Everything is displayed in a bi-dimensional interactive way, showing the cursor location in the working coordinates and including panning, zoom and view-all options. The crossing-crossing application displays matched sets of nodes (crossings), with the same color in both networks. The link-link application tool also allows visualization of the capacity (or free speed) property value of both networks' links, using a color scale, as shown in Figure 9.3.
- Selection: The applications enable selection of links and nodes from both networks. The crossing-crossing option allows only selection of node sets. The link-link application allows selection of links' sequence. This can be done directly, or by selecting an origin node, a destination node and running a "select shortest path algorithm tool". It is also possible to select the other link instead of the first one chosen.
- Matching and Deletion: The applications allow creation of a similarity relationship between elements selected in both networks, sets of nodes, or sequences of links.
- Saving: The applications allow located relationships to be saved.
- Loading: The applications allow the loading of previously located relationships.
- Others: The crossing-crossing application executes and automatically verifies currently found matching, to avoid repeated nodes. It also enables clearing of the current selection. The link-link application allows automatic navigation to a link, or node, specified by the user, using its ID. It also enables the undoing of previous matching.

9.1.3 Results

All low-res network links were matched to high-res links, updating the corresponding link properties. Figure 9.4 shows the differences in travel capacities between original navigation network values and the final version. Eight hours of manual work were required to match crossings and ten hours of manual work to match links. Obviously, improvements in accuracy and completeness of the automatic matching algorithms reduce the manual work time.

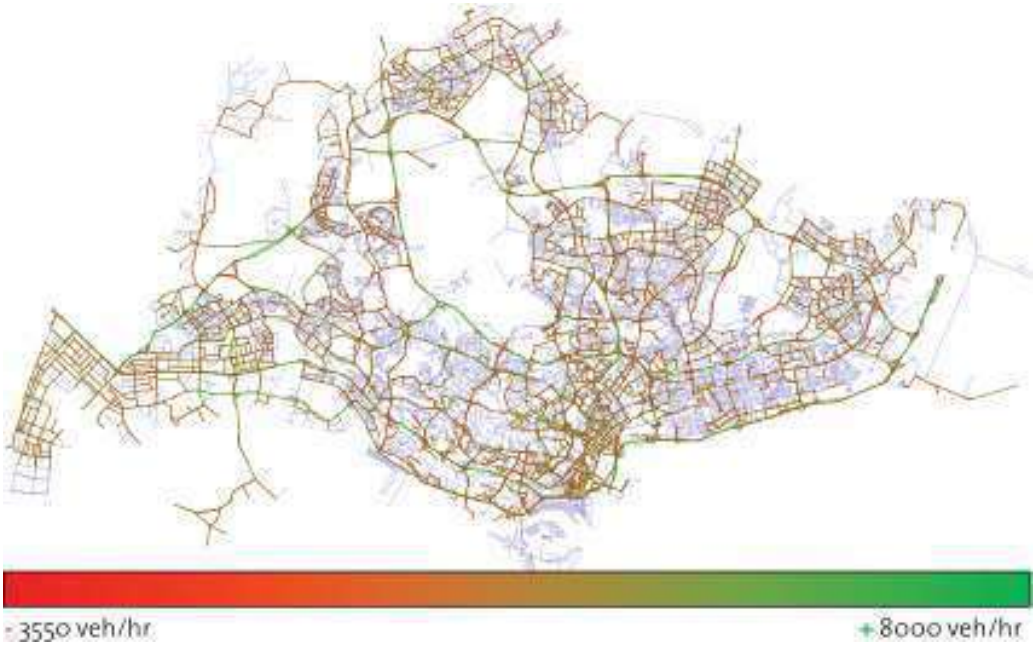


Figure 9.4: Resulting changes in navigation network travel capacity property.

CHAPTER 10

The "Network Editor" Contribution

Kai Nagel

10.1 Basic Information

Entry point to documentation:

`http://matsim.org/extensions` → networkEditor

Invoking the module:

`http://matsim.org/javadoc` → networkEditor → `RunNetworkEditor` class

Selected publications:

none

10.2 Short Description

This is, beyond the two network editors described in Chapters 8 and 9, a third network editor. It is older than the other two and has not been systematically maintained, but it still seems to be working and so it is still there.

How to cite this book chapter:

Nagel, K. 2016. The "Network Editor" Contribution. In: Horni, A, Nagel, K and Axhausen, K W. (eds.) *The Multi-Agent Transport Simulation MATSim*, Pp. 73–74. London: Ubiquity Press. DOI: http://dx.doi.org/10.5334/baw.10.

SUBPART TWO

Mobsim

CHAPTER 11

QSim

Marcel Rieser, Kai Nagel and Andreas Horni

11.1 Vehicle Types and Vehicles

For a variety of reasons—e.g., vehicle-specific emissions calculations (Chapter 36), 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 would then have to know which vehicle it wants to use. This is indeed possible with the planned vehicle ID that MATSim route objects can store. This functionality is switched on by first loading an additional vehicles file (see Section 6.6) and then configuring the QSim as

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

(available with 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.

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

How to cite this book chapter:
Rieser, M, Nagel, K and Horni, A. 2016. QSim. In: Horni, A, Nagel, K and Axhausen, K W. (eds.) *The Multi-Agent Transport Simulation MATSim*, Pp. 77–80. London: Ubiquity Press. DOI: http://dx.doi.org/10.5334/baw.11.

```
<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 Sources 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.

At the moment, alternative approaches to vehicle generation need to be programmed as script-in-Java. See, e.g., `RunMobsimWithMultipleModeVehiclesExample` under `http://matsim.org/javadoc` → main distribution for a reference to a script that generates mode-specific typical vehicles for each mode. Simulation experiments using this feature have been performed for the Patna scenario as reported in Chapter 77.

Vehicle Behavior Vehicles need to be available where they are needed. It is, for example, impossible 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. The idea here is that such synthetic travelers should have within-day replanning strategies (see Chapter 30) to cope with unexpectedly unavailable vehicles; any attempt to use an unavailable vehicle points to an error in the driver's behavioral logic.

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 the car sharing extension (Chapter 22).

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

This option can be enabled by using

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

in the `qsim` section of the config file.

11.2 Other

The simulation is able to handle time-variant networks (Lämmel et al., 2010), within-day replanning (Dobler, 2009, see Chapter 30) and traffic lights (Neumann, 2008; Grether et al., 2011b, 2012, see Chapter 12). An earlier multimodal approach, targeted at overcoming the teleportation estimates of non-motorized modes, and particularly focused on pedestrians, is presented in Chapter 21.

SUBPART THREE

Individual Car Traffic

CHAPTER 12

Traffic Signals and Lanes

Dominik Grether and Theresa Thunig

12.1 Basic Information

Entry point to documentation:

`http://matsim.org/extensions` → signals

Invoking the module:

`http://matsim.org/javadoc` → signals → `RunSignalSystemsExample` class

Selected publications:

Grether et al. (2011a); Grether (2014)

12.2 Motivation

Traffic signals ensure security of travelers at junctions and regulate right of way. Furthermore, by assigning green times to the different approaches of a junction, they determine and evaluate junctions' performance. There are different strategies for traffic signal control: fixed-time traffic signal control, for example, periodically repeats the same schedule for signalization, while traffic-responsive signal control reacts dynamically to the prevailing traffic patterns to improve the junction or system performance. Traffic signal control can improve the traffic conditions at a single junction, but the whole system can be worse if a single junction is improved. Hu and Mahmassani (1997) argue that second order or network effects should be taken into account when effects of signal control strategies are tested. Network effects include drivers' reactions: not only route choice, but also scheduling. Thus, traffic control, especially traffic-responsive signals, need certain constraints. Otherwise, traffic may become unstable: rapidly at two nearby junctions, or at the network level (Lämmer and Helbing, 2010). MATSim can capture most of these effects. This chapter reviews

How to cite this book chapter:

Grether, D and Thunig, T. 2016. Traffic Signals and Lanes. In: Horni, A, Nagel, K and Axhausen, K W. (eds.) *The Multi-Agent Transport Simulation MATSim*, Pp. 83–88. London: Ubiquity Press. DOI: http://dx.doi.org/10.5334/baw.12.

concepts, usage and restrictions of the traffic signal control extension for MATSim. The chapter is particularly interesting for MATSim users, who plan to simulate traffic signals microscopically. If one wishes to capture signalization effects on a rather coarse level, consider the approach presented in Charypar (2008, pp. 139), that can be realized with the time variant network feature of MATSim (Lämmel et al., 2010). Before we go into detail on motivating traffic signals with MATSim, a case study is reviewed.

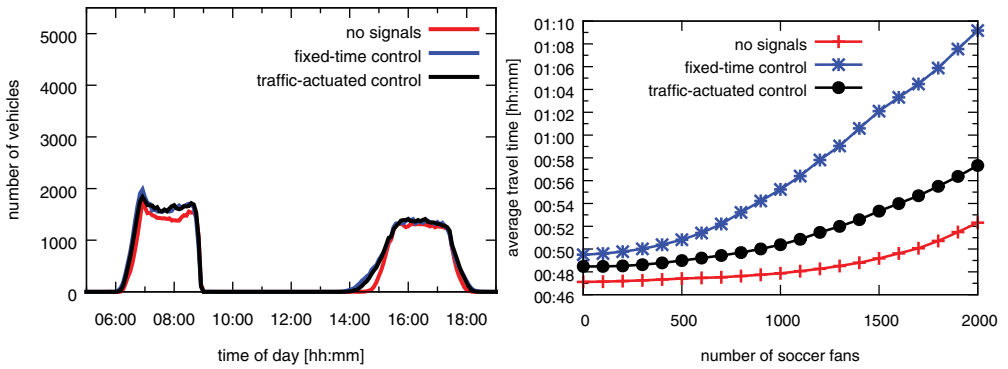
12.2.1 Case Study

The Cottbus scenario presented in Chapter 66 is applied to illustrate the influence of traffic signal control. This section summarizes results published in Grether et al. (2011a); Grether (2014). Readers interested in details are referred to these publications.

The runs sequence is performed with three different signal control strategies: In a first simulation sequence, all traffic signals are switched off. This can be used as a lower bound for results of signal control, since it assumes that vehicles are able to traverse a crossing without an accident, i.e., they are able to drive "through each other". The next sequence uses the fixed-time setup. In the third and final, sequence, all traffic signals are controlled by a traffic-actuated stage length control. The control is based on pre-timed, fixed-time schedules. The green times of the fixed-time schedules are reduced to a minimal green time of 5/10 seconds. If vehicles are still approaching at the end of this reduced green time, it is extended up to a predefined maximum.

Simulation results for iteration 1 000 of the Cottbus commuter scenario are depicted in Figure 12.1(a). The number of vehicles simultaneously on the road is plotted over the time-of-day. The results are quite similar for all signal control strategies; differences are small because of the lack of heavy congestion in the Cottbus scenario.

A change of signal control has more effect if unexpected traffic occurs in the network. It is assumed that the local soccer club, "FC Energie Cottbus", has a tournament taking place on a normal weekday, interfering with regular commuter traffic. In iteration 1 000 of the commuter scenario, in addition to the commuters 0 to 2 000 vehicles drive to the Cottbus soccer stadium during the evening peak. It is assumed that 25 % of these fans come from Cottbus, while the other 75 %



(a) No vs. fixed-time vs. traffic-actuated signal control, commuter traffic, iteration 1 000.

(b) Average travel time for unexpected event traffic, iteration 1 000.

Figure 12.1: Simulation results for the Cottbus traffic signal scenario: The simulated change of traffic signal control results in small travel pattern changes in the relatively quiet commuter scenario (left). If unexpected traffic occurs on the network, the traffic-actuated signal control enables travel time savings (right).
Source: Grether (2014)

come from the "Spree-Neiße" area around Cottbus, and that all fans start their trips between 5 pm and 6 pm.

Figure 12.1(b) plots the number of soccer fans on the x-axis, and the average travel time of all travelers on the y-axis. Without any additional vehicles, the traffic-actuated signal control leads to a gain of approximately 1 minute per traveler. The more additional traffic approaches the stadium, the more the traffic-actuated control saves travel time. When 2 000 additional vehicles are on the road, travel time savings reach approximately 15 minutes per traveler.

Summarizing: Slightly jammed commuter scenarios, where a change in traffic signal control leads to noticeably decreased overall travel time, have not yet been simulated with MATSim. Looking at different objectives with more fine-grained analysis tools can reveal network wide effects (e.g., see the analysis using macroscopic fundamental diagrams Grether, 2014, pp.114), but this is work in progress. More heavily jammed scenarios can increase the overall traffic impact of a change in traffic signal control. Nevertheless, the case study shows significant effects of traffic-responsive signal control when something unexpected happens and travelers do not react.

12.2.2 Overview MATSim & Traffic Signals

This case study highlights some previously researched MATSim traffic signals simulations aspects. MATSim is not always the traffic signal control "tool of choice" for all questions. The code base, however, can help simulate other use cases, e.g., evacuation or air transport scenarios; MATSim's open source nature provides hooks and interfaces for extension. But one must consider the amount of work required, the current state of development and specific project planning. The rest of the chapter goes into more detail. Section 12.3 provides some traffic signal control background, vocabulary, and options for modeling traffic signals with MATSim. Technical details can be found in the traffic signals user guide. Section 12.4 goes into details on network and traffic flow modeling. Iterations and learning are discussed in Section 12.5. When it comes to agent based learning, MATSim is very fast—the presented case study requires, on average, 17 seconds computation time per iteration—for scoring, replanning, and output. One complete run sequence: (1 000 iterations, single core mobility simulation, multi-core replanning) was simulated in 9 hours and 12 minutes. The simulation speed allows exploration of network-wide behavioral reactions to traffic signal control changes and the resource efficient simulation enables the joint simulation of several policies. Before publishing results, one should consider several specific aspects of evaluation and simulation results interpretation. Hints are provided in the conclusion, Section 12.6.

12.3 Traffic Signal Control

On a coarse level, control strategies for traffic signals can be classified in fixed-time and traffic-responsive strategies.

Fixed-time traffic signal control periodically assigns green times for each junction approach. Cycle time and green split are not modified within short time periods. To establish green waves between adjacent junctions, the green light start for approaches within the cycle can be adjusted by a global timer; these shifts are referred to as (coordination) offsets. For optimization of fixed-time signals, different equilibrium traffic flow regimes are determined for several periods of time, e.g., weekday morning, midday, evening and night plus a separate estimate for weekends. Optimization may target all signalized junction parameters—green split, cycle, offsets, and phase composition, but it is not possible to react to current changes in equilibrium traffic flows.

Traffic-responsive control reacts to current traffic patterns, adjusting traffic signal control parameters on the fly. In principle, all available information on prevailing traffic patterns can be used. The diversity of traffic-responsive control algorithms is wide; for a review, see Grether (2014).

MATSim's traffic signal module is designed to simulate every traffic signal control strategy. The module provides a default implementation for fixed-time control. Traffic-responsive strategies require custom implementation of the control algorithm, but can use existing data formats and fixed-time control infrastructure. Data is divided into five different types of input:

Signals & Systems: The location of the traffic signal hardware on the network is usually independent from the control strategy. Signals can be located at the end of a link or a lane (see the next section for further discussion of lanes). Signals are attached to a system that reflects, e.g., all signals of a junction or even larger units. Each signal system is controlled by exactly one control algorithm at a time.

Signal Groups: Traffic signals must be attached to a group. A group of signals shows the same color at the same time. Each time a signal group changes its state, a MATSim event is triggered. There is no explicit phase representation; if required, this can be realized over signal groups.

Signal Control: Specifies the control algorithm for each signal system. Data comprises information for fixed-time control and can be extended to capture custom control algorithms' parameters.

Amber: Specifies the amber phase at the beginning and end of green time. Currently, driving is not permitted if a traffic signal group shows amber light and this information is used only for visualization purposes.

Intergreens: The inter-green time specifies minimal time period between the ending of one and beginning of another signal group's green time. This information is important because MATSim's traffic flow model does not contain any collision detection. A validation module reads the event stream and triggers a warning, or an error, if security constraints are violated. Further, customized control strategies can access this information to ensure security aspects' control validity.

For detailed information on file structures and how to link them in the MATSim config file, we refer to the user guide in the contribution "signals".

The next section explains network representation and traffic signal location in more detail.

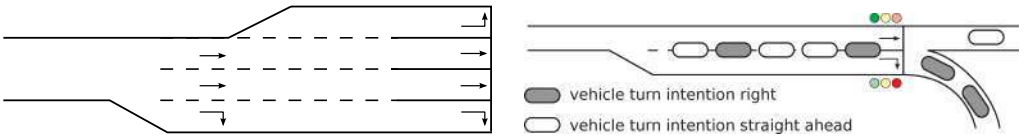
12.4 Network Representation & Traffic Flow

This section explains transport network representation with microscopically modeled traffic signals. In MATSim, transport network representation is a static, directed graph, consisting of nodes and links. Links depict road segments, while nodes can be interpreted as decision points in space with a coordinate as attribute, but no spatial dimension.

Figure 12.2(a) illustrates a typical layout of a real-world road segment, with several turn pockets at its end. If the whole road segment is modeled as a single link with MATSim's queue model, the first vehicle stopping at a red traffic signal at the end of this link will block all other vehicles approaching upstream, see Figure 12.2(b). In respect to the road layout shown in Figure 12.2(a), this is unrealistic. Figure 12.3(a) sketches the network layout for a more realistic modeling. Vehicles with distinct turn intentions do not block each other until the available space for queuing on the turn pocket is used completely, see Figure 12.3(b).

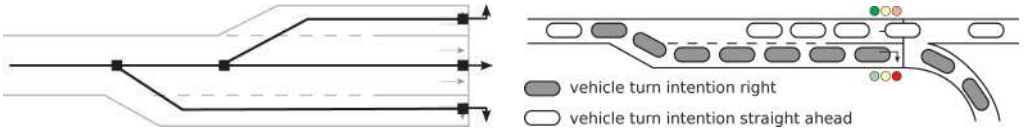
In principle, one can model each turn pocket as a link and put traffic signals at its end; but considering overall project constraints, this has implications for network modeling and routing.

In MATSim, all domain-relevant attributes differing from geospatial location, e.g., traffic count data, transit stops, transit lines, or speed limits, are attached to links. If one of this attributes changes, one must model several links. Frequently, geospatial location of such attributes is insufficient for a fully automatic matching of attributes to links; some data requires manual



(a) Typical real road layout. (b) Single queue, spill-back is not captured correctly.

Figure 12.2: Transition from a real road segment to a graph layout with a single queue: the missing turn pockets representation prevents vehicles passing each other and cannot capture the traffic signal control for different turning moves.
Source: Grether et al. (2012)



(a) Part of the graph required to model the road layout. (b) Multiple queues, spill-back is captured correctly.

Figure 12.3: Transition from a real road segment to a graph layout with multiple queues: each turn pocket is represented by its own queue. Traffic signal control for different turning moves is captured; vehicles can pass each other, unless the queue spills over.
Source: Grether et al. (2012)

post-processing. To simulate traffic signals and turn pockets with an already existing scenario, carefully consider the matching process before changing the network.

Travelers' routes are specified by link sequences within MATSim and routes are generated by a shortest path algorithm requiring a cost function for links. In standard MATSim, link travel time is part of a link's cost. When modeling turn pockets as links, the shortest path algorithm is responsible for selecting the appropriate turn pocket on a route. If modeling includes turn restrictions, ensure that they are captured by the shortest path algorithm and note that the required number of iterations increases if many turn pockets lead to the same downstream link. It is important to understand route generation and network modeling interaction when modeling turn pockets as links.

If network modeling or routing issues clash with other project goals, there is an alternative. MATSim allows the modeling of a subgraph on top of each link to reflect the structure shown in Figure 12.3(a). The links of the subgraph are then called *lanes*. At the beginning of a link, only one lane can be modeled; at the end of a link, different lanes can exist to model turn pockets. A vehicle must be in the correct turning lane for the next downstream link of its route. If there is only one lane towards the downstream link, the vehicle uses this lane. If there is more than one lane leading to the next downstream link, the vehicle is placed on the lane currently containing the fewest other vehicles. Using lanes, specific turning moves can be forbidden because the shortest path algorithm underlying network graph is modified; thus, turn restrictions are considered when the network graph is created. The shortest path calculation captures the effects of lanes without further modification (see Grether, 2014, pp. 21).

As well the differences mentioned above, lanes exhibit behavior similar or equal to links. Vehicles entering or leaving lanes trigger events with the same structure and information as link enter and leave events. Traffic signals can be placed at the end of links and lanes. Traffic on each lane is

simulated the same way as for links. Traffic flow increase is linear in a signal's green time for both links and lanes.

The decision to use or not use lanes is arbitrary. Most MATSim scenarios with signals are set up using lanes; the code base is well debugged. Without lanes, the code for traffic signals is also tested; one should check carefully for artifacts and understand influences on route generation.

12.5 Iterations & Learning

This section discusses interaction between traffic signals and travelers within the MATSim iteration cycle.

Meneguzzer (1997) defines the combined traffic assignment and control problem as finding a tuple (f^*, g^*) of traffic flows f and signal settings g under policy P that fulfills

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

where f^e is a function mapping signal settings to equilibrium traffic flows and g^P a function mapping traffic flows to signal settings under policy P. The formulation neatly shows the mutual interaction of traffic patterns and signal settings. The formulations do not capture the time horizon where these interactions take place.

Traffic signal interpretation within the MATSim iteration cycle depends strongly on signal control type and learning mechanism interpretation. For fixed-time control, the fixed-point interpretation can be valid, at least if one does not anticipate unexpected events on the demand side. For traffic-actuated signal control strategies, no standard interpretation can be provided. Readers seeking more detail are referred to Grether (2014, pp. 75). We conclude with this advice; clearly document what and how was simulated and provide an interpretation that makes sense for each individual project.

12.6 Conclusion

MATSim can simulate traffic signal control microscopically. However, certain traffic signal effects are not represented by MATSim without further customization and implementation, e.g., microscopic deceleration and acceleration as a reaction to traffic control. Evaluations must be checked and interpreted against the simulation setup to ensure that everything derived from simulation results is also appropriately simulated. This chapter provides an overview of traffic signals in MATSim, detailing what to consider before taking first steps in larger scenarios. Details for implementation can be found in the javadoc documentation referenced above. For the detailed scientific discussion of modeling aspects the reader is referred to Grether (2014).

We think that MATSim is a superior tool for microscopic simulated traffic-responsive signal control that should be analyzed network-wide, assuming heterogeneous user reactions.

CHAPTER 13

Parking

Rashid A. Waraich

13.1 Basic Information

Entry point to documentation:

`http://matsim.org/extensions` → parking

Invoking the module:

`http://matsim.org/javadoc` → parking → `RunParkingExample` class

Selected publications:

Waraich and Axhausen (2012); Waraich et al. (2013a); Waraich (2014); Waraich et al. (2014b)

13.2 Introduction

The MATSim simulation, by default, does not consider parking infrastructure or supply constraints. However, this can lead to artificially high car traffic to city centers in the model, often not the case in the real world, due to limited parking. The modeling of parking is also important because traffic-related policies can be designed around parking; e.g., raising prices for parking at certain times of the day, or reducing parking supply in an area, can impact travel demand.

This chapter describes work done to bridge this gap via parking models for MATSim .

13.3 Models

For technical reasons, parking modeling efforts in MATSim were divided in two parts: parking choice and parking search, described in the following two subsections.

How to cite this book chapter:

Waraich, R A. 2016. Parking. In: Horni, A, Nagel, K and Axhausen, K W. (eds.) *The Multi-Agent Transport Simulation MATSim*, Pp. 89–92. London: Ubiquity Press. DOI: http://dx.doi.org/10.5334/baw.13.

13.3.1 Parking Choice Model

The first approach for modeling did not change the MATSim traffic simulation; it extended it to capture parking supply through `controler listeners` and event handling. This means that no rerouting due to parking took place during the simulation. However, changed routes could be incorporated in a post-processing step, as described in Waraich and Axhausen (2012).

In the most general case, a parking choice model performed the following simulation steps; when a vehicle arrived at a destination in MATSim, the parking choice model assigned a parking spot in the agent's area, according to a customizable algorithm (e.g., utility maximization). The assigned parking place was marked as occupied on arrival and became unoccupied again when the agent departed, allowing the model to simulate supply side constraints with the same temporal resolution as the basic MATSim model.

A simple parking choice model version was able to consider only walk distance minimization, ignoring other user preferences and park at the closest available public parking. A simple model like this was able to partially solve one of the main problems of the un-constrained parking model in MATSim; it made an area with little parking less attractive as a car destination due to longer walk distances. Parking model integration with MATSim was achieved by adding a term for the parking operation to the agent's overall plan scoring function, as follows:

$$S_{parking} = S_{walking} + S_{parking\ costs} + S_{parking\ search\ time} \tag{13.1}$$

Beyond walking distance disutility, this scoring function could also include additional features like cost, or even estimated parking search times, using models like Horni et al. (2013a).

A Zürich city study, which implemented a parking choice model and included trade-off between walk distance and parking cost, was presented in Waraich and Axhausen (2012). This study also distinguished between public, private and reserved parking, where only certain people (e.g., disabled) or certain vehicles could park (e.g., electric vehicles). Figure 13.1 shows parking choice models employed in this study, where a distinction between public, private and reserved parking was made. In Waraich et al. (2013c), another study for modeling parking in MATSim was reviewed, exploring individual gender and age parking preferences. Utility function parameters used in this study were based on a stated preference survey in Switzerland.

13.3.2 Parking Search

The parking choice model presented in the previous section could capture many relevant aspects of parking. However, it did not model parking search behavior; studies conducted around the world suggest that, on average, around 30 % of city centers traffic could be due to parking search traffic Shoup (2004). Thus, it seems extremely important to capture parking search related traffic in transportation models.

A first idea about model parking search traffic in MATSim was presented in Waraich et al. (2012). The basic idea came from surveys suggesting that people select certain strategies they think will be beneficial for them when starting the parking search process (Axhausen and Polak, 1989). Proof of this concept for development was attempted, using within-day replanning (see Chapter 30 and Dobler et al. (2012)). However, this path was aborted after development of several initial strategies, where performance and integration issues led to dead ends (Waraich et al., 2013c); performance after optimization was around 24 times slower than the original runs without parking operations.

An alternative path closer to the idea presented in Waraich et al. (2012) was successfully attempted, using a JDEQSim based model (see Section 4.3.2) with within-day support and travel time approximation, as seen in PSim (see Chapter 39, Fourie et al. (2013)). This removed overhead,

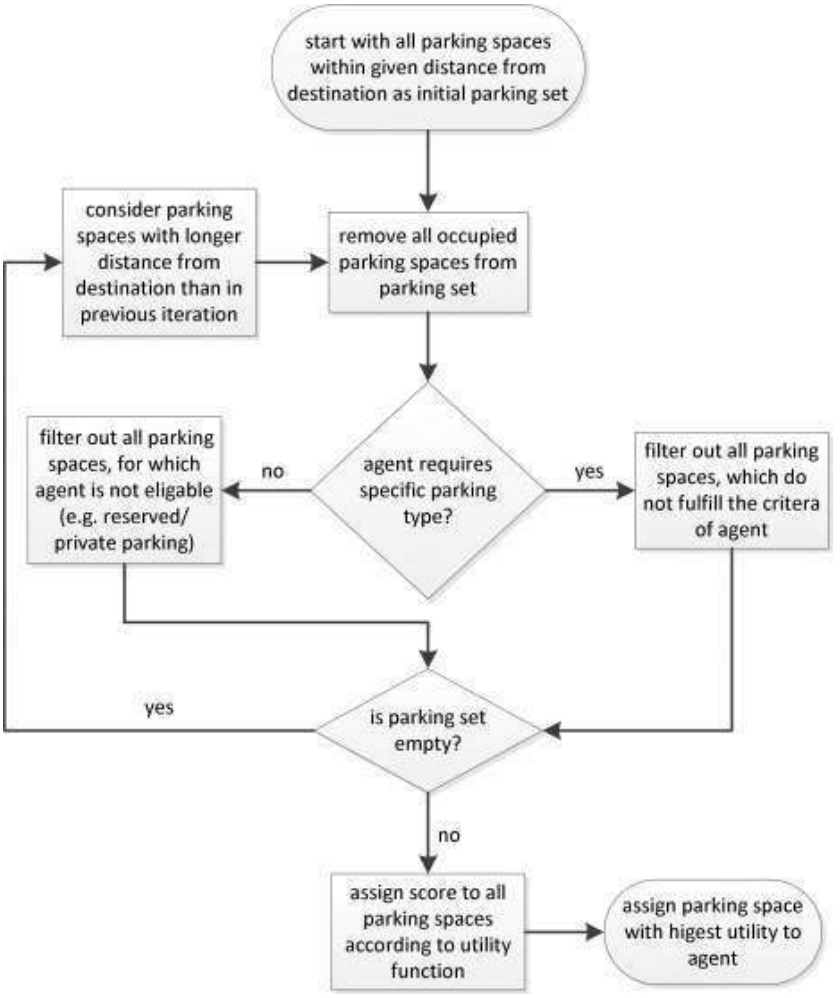


Figure 13.1: Parking choice algorithm.

Source: Waraich and Axhausen (2012)

present in the previous approach, enabling flexibility to implement many of the parking strategies presented in Axhausen and Polak (1989) and beyond. Publication of this approach's first results are expected in 2015.

Unfortunately, the approach is not available in packaged form to other users of MATSim.

13.4 Applications

Clearly, the parking model applications presented were important, diverse and especially well-suited for policy design; one example of traffic policy design by means of targeted reduction of parking supply was presented in Waraich and Axhausen (2012). Waraich et al. (2013c) explained an application of performance-based pricing for parking in MATSim, where iteratively parking prices were adapted to match demand. An integration of parking choice and electric vehicle charging was presented in Waraich et al. (2014a) for a Zürich case study and Bemetz and Hohenfellner (2014)

described an even more sophisticated test model for parking and EV (Electric Vehicle) charging, with various types of charging speed and prices.

13.5 Usage

A general parking choice model was included in the parking contribution of MATSim, which provided various extension interfaces; examples were included in the parking contribution to provide help with extension.

CHAPTER 14

Electric Vehicles

Rashid A. Waraich and Joschka Bischoff

Entry point to documentation:

`http://matsim.org/extensions` → transEnergySim

Invoking the module:

No predefined invocation. Starting point(s) under `http://matsim.org/javadoc` → transEnergySim → `RunTransEnergySimExample` class.

Selected publications:

Waraich et al. (2013d); Galus et al. (2009, 2012b); Waraich (2013); Galus et al. (2012a); Waraich (2012a,b); Waraich et al. (2014a); Waraich and Axhausen (2013)

14.1 Introduction

Research related to EV modeling in MATSim started in 2008/2009, with an electricity grids project (Waraich et al., 2013d); it's goal was to uncover potential bottlenecks and/or constraint violations in Zürich city's lower voltage grid due to future EV charging. A framework emerged from the research for EV modeling, called TESF (Transportation Energy Simulation Framework) (Waraich et al., 2014a). This resulted in various framework extensions and enabled simulation of various scenarios (Waraich et al., 2014a; Waraich, 2013; Abedin and Waraich, 2014; Schieffer, 2011; Galus and Andersson, 2011; Galus et al., 2012a; Bischoff, 2013; Bischoff and Maciejewski, 2014). This chapter provides advice on these research directions and serves as a starting point for modeling EVs in MATSim.

14.2 Models

The main reason for modeling EVs in TESF was simple: it was essential to keep track of the battery charging state in the EVs. This meant that, as the EV was driving, depletion of the

How to cite this book chapter:

Waraich, R A and Bischoff, J. 2016. Electric Vehicles. In: Horni, A, Nagel, K and Axhausen, K W. (eds.) *The Multi-Agent Transport Simulation MATSim*, Pp. 93–96. London: Ubiquity Press. DOI: http://dx.doi.org/10.5334/baw.14.

batteries was simulated. It was also important to consider the charging process of EVs at charging infrastructures.

While the basic EV modeling mechanisms were simple, there were many details to ponder when modeling scenarios. The TESF framework provided both interfaces and implementations to cope with more complex cases, e.g., defining a vehicle that can charge without contact while driving, for example, by using dynamic inductive charging. Furthermore, charging mechanisms themselves could also be quite complex. The following sections provided some details on this, as well as different models involved.

14.2.1 Energy Consumption Model

When a vehicle was defined in TESF, it could be assigned an energy consumption model, defining how much energy the vehicle used while driving. For conventional vehicles, just energy consumption could be logged using such a model; however, for electric vehicles, the energy consumption model was used to update the on-board battery system state of charge. PHEVs (Plugin Hybrid Electric Vehicles) can use both electricity and gasoline for driving and therefore had two different energy consumption models assigned to them for modeling these two modes. When this was written, a series hybrid model were implemented in TESF (Chan, 2007), which used electricity as long as the battery charge state was above a certain threshold value, then switched to gasoline. This type of vehicle could also be charged using a plug, like a battery electric vehicle. For PHEVs, car manufacturers often defined rules governing when a vehicle should switch between battery and gasoline use. The TESF framework provided interfaces and examples of how more advanced control strategies for PHEVs could be implemented.

14.2.2 Charging Infrastructure

In addition to plug-based charging, inductive charging infrastructure was also modeled in TESF, with two types: dynamic and stationary. The dynamic inductive charging infrastructure was often embedded in roads; vehicles able to use such infrastructure could charge while they drove. Stationary inductive charging was, more or less, modeled like plug charging; however, charging interfaces between vehicle and the charging infrastructure had to match for the charging process to function.

Another fast route to a full battery was to replace/swap the used battery for a new one at a specialized infrastructure, sometimes referred to as a swapping station (Li et al., 2011). A basic modeling of this approach was provided in TESF, which could be extended and detailed further according to specific scenario needs.

14.2.3 Charging Schemes

When an EV connected to any infrastructure for charging, a scheme was needed to define how the vehicle charging would operate; should the vehicle start charging immediately, or would charging depend on an agent's pricing preferences, which could vary with time and location? Negotiations between the vehicle computer and grid operator were also possible, which perhaps allowed for some electricity grid temporal flexibility, while fully charging a vehicle's battery before departure (sometimes referred to as "smart charging"). Various charging schemes were part of the TESF and were be used to model other more complex charging schemes; TESF-simulated examples of various charging schemes can be found in Waraich et al. (2013d).

14.2.4 Vehicle-to-Grid

When studying electric vehicles, charging is not the only topic of interest; V2G (Vehicle-to-Grid) applications where electric vehicle batteries supply power and energy back to the grid (Kempton

and Tomic, 2005) were analyzed. While the integration of V2G models for MATSim was limited at any given time, an application related to V2G and intermittent energy generation at wind parks using MATSim can be found in Galus and Andersson (2011) and a preliminary attempt to integrate V2G in TESF was described in Waraich et al. (2014a); Schieffer (2011).

14.2.5 Vehicle Choice

When conducting electric vehicle studies, each vehicle owner usually has to be assigned a specific type: e.g., electric vehicle, conventional vehicle, plug-in hybrid, etc. Sometimes, these assignments were random, while ensuring vehicle type share constraints for the scenario (e.g., Waraich et al., 2014a). Often, however, possible financial or infrastructural incentive implications, e.g., different toll prices, parking fees or fuel prices for different vehicle types, had to be evaluated. A replanning module for vehicle choice, also covering EVs, was recently implemented; first results should published soon and can also be integrated in TESF.

This section provided an overview of the various TESF framework parts and the following section an application of a TESF contribution, that modeled electric taxis.

14.3 Application: Electric Taxis

Combination and extension of both the TESF and VRP (Vehicle Routing Problem) contribution (see Chapter 23) allows simulation of BEVs (Battery Electric Vehicles) taxi fleets. For electric vehicles, vehicle charging process was adapted; for taxis, the concept of taxi ranks and a modified optimizer sending idling taxis to the rank and only dispatching vehicles with sufficient battery charge were introduced.

14.3.1 Taxi Ranks

After dropping off passengers, taxis proceeded to the nearest rank location, unless there was an immediate follow-up request. Queuing took place at the rank location; the taxi that arrived first would leave the rank first. Other types of queuing were also tested, e.g., a dispatch by battery SOC (State of Charge). Ranks were not mandatory; however, driving there between trips would be typical German taxi driver behavior.

14.3.2 Charging Process

Chargers could be located at taxi ranks or any other link. Following any given BEV `AgentArrivalEvent` at a charger location link, charging would begin if

- there was a free charging spot,
- the vehicle's SOC was under a certain threshold,
- at least two minutes of time passed required for parking the car and plugging it in.

Electric taxi simulation has been used in Mielec, Poland (Bischoff, 2013; Bischoff and Maciejewski, 2014). When this was written, an application for Berlin was in progress.

14.4 Usage

The TESF contribution contained many features described above and interfaces were provided for framework extension. Examples were also given for the setup of different scenarios: e.g., energy consumption model, vehicle types, charging schemes, etc.

CHAPTER 15

Road Pricing

Kai Nagel

15.1 Basic Information

Entry point to documentation:

`http://matsim.org/extensions` → roadpricing

Invoking the module:

`http://matsim.org/javadoc` → roadpricing → `RunRoadPricingExample` class

Selected publications:

Rieser et al. (2007a, 2008); Grether et al. (2008)

15.2 Introduction

Roadpricing is a controversial policy measure (e.g., Button and Verhoef, 1998). Its implementation in MATSim is conceptually straightforward (Rieser et al., 2007a, 2008; Grether et al., 2008): Essentially, for each vehicle entering a link at a given time, the appropriate toll is computed and charged to the vehicle's driver. The scoring function will pick this up by the term (see Equation (3.4))

$$S_{trav,\,car,\,q} = \ldots + \beta_m \cdot \tau + \ldots ,$$

where τ is change in the monetary budget invoked by all toll payments (usually negative) and β_m is the marginal utility of money (also see Chapter 3 and Chapter 51). The driver then takes this into account making decisions, e.g., for route choice, departure time choice, mode choice, destination choice, etc., and then trades off toll payments with other elements of his or her scoring function.

How to cite this book chapter:

Nagel. K. 2016. Road Pricing. In: Horni, A, Nagel, K and Axhausen, K W. (eds.) *The Multi-Agent Transport Simulation MATSim*, Pp. 97–102. London: Ubiquity Press. DOI: http://dx.doi.org/10.5334/baw.15.

It should be clear that this automatically picks up all kinds of heterogeneities, for example:

- Traveling at a different time may lead to a different toll, but possibly also to different schedule delay costs (Section 3.2.5).
- Different vehicle types may be charged different tolls (Kickhöfer and Nagel, 2013).
- Different travelers may have different time values (Nagel et al., 2014), which may even vary according to the time of day.

However, one challenge is that the innovative modules (Section 4.5) must be consistent with the scoring now modified by road pricing. The approach just described will not work if, for example, the router consistently generates toll-avoiding routes for a synthetic person with a high time value, who would normally wish to pay for a faster option. In a case like this, if a suitable route is never generated, the scoring cannot identify it, giving the choice process no chance to select it in subsequent iterations.

However, processing every detail for each individual, i.e., not only the marginal utility of money, but also specific time pressure at the route search time, is quite complex.

An alternative approach is to make the router *randomizing*, i.e., to run it with a randomly generated time value every time necessary for a given person. Computational experiments with this approach produce solutions for synthetic travelers approximately as good, or even better, than an "engineered" router (Nagel et al., 2014). At the same time, the software consistency burden is significantly reduced, noticeable in the smaller amount of information to be extracted from the agent during each router call.

15.3 Some Results

15.3.1 Effect of an Afternoon Toll on Morning Traffic

In a first demonstration of capabilities, an afternoon toll for the Zürich area was simulated. While this is an unlikely policy scheme, it still clearly demonstrated the advantage of the integrated approach over other approaches. Not only did the synthetic travelers switch to public transit, but they also did so for the morning rush hour, where no toll was charged (Figure 15.1). Thus, the MATSim approach proved its ability to affect the whole daily plan, not just the trip. For more information, see Rieser et al. (2008).

15.3.2 Income-Dependent Values of Time

Similar to Rieser et al. (2008), Kickhöfer et al. (2010); Kickhöfer (2014) introduced a distance-based morning peak toll on the same links between 6:30 am and 9 am. Toll levels were incrementally increased from 0.28 CHF/km up to an almost prohibitive price of 44.80 CHF/km. The studies assume income-dependent utility functions with a decreasing marginal utility of money. The goal was to (i) identify the welfare-maximizing (see e.g., Tirachini et al., 2014, Section 2.5) toll level, which is potentially dependent on the aggregation rule of user benefits (see Chapter 51), and (ii) to investigate distributional aspects of such pricing schemes. The studies showed that changes in travel patterns resulting from the morning peak toll impacted the whole day, affecting traffic patterns in the afternoon. Furthermore, the study showed that such a parametric approach is capable of identifying the welfare-maximizing toll level. However, results also indicated that the overall welfare effect level depends strongly on the aggregation rule for user benefits, i.e., if one first monetizes

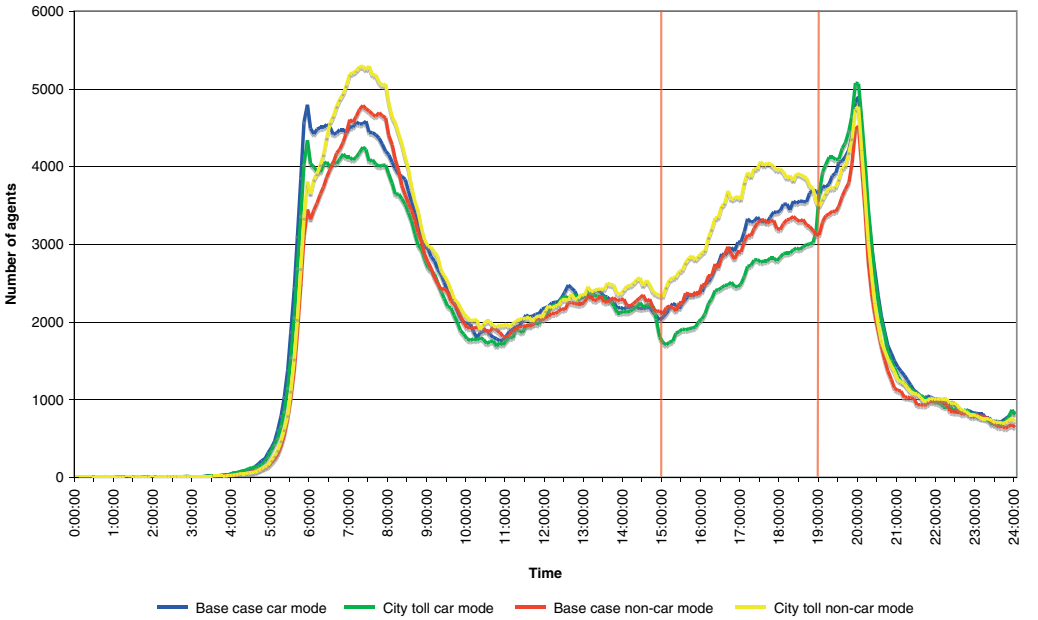


Figure 15.1: An afternoon city toll (between 3 pm and 7 pm) affects mode choice not just during the toll time, but also in the morning.

Source: Rieser et al. (2008)

individual utilities and then adds up, or first adds up utilities and then monetizes. Even the sign of that effect might not be stable depending on that choice. For more information, please refer to the two studies above.

15.3.3 Integrated Passenger and Freight Toll Simulation for the Gauteng Province in South Africa

A large scale application was undertaken for the Gauteng province in South Africa (Chapter 69). It is based on the so-called e-toll, which was switched on in December 2013. The e-toll should, logically, charge different rates for different vehicle types, with higher rates for heavy trucks. Again, logically, this should go along with higher time values of the driver-vehicle-units. Somewhat surprisingly, this turned out to be difficult to do with the MATSim software structure in place when the project was started in 2008. While it was easy to charge the freight vehicles a higher toll, it was difficult to give different replanning methods and different scoring function to the freight population; it was essentially impossible to feed the router with different time values for the freight population. This was an important driver for much development in recent years, including making the scoring function more accessible (Section 45.2.10), allowing different replanning strategies for different sub-populations (Section 4.5), and reducing consistency requirements between the router, the vehicle-based toll and the driver-based scoring function (Nagel et al., 2014).

The simulation, as expected, predicts reduced traffic volumes on the tolled roads and increased volumes elsewhere (Figure 15.2).

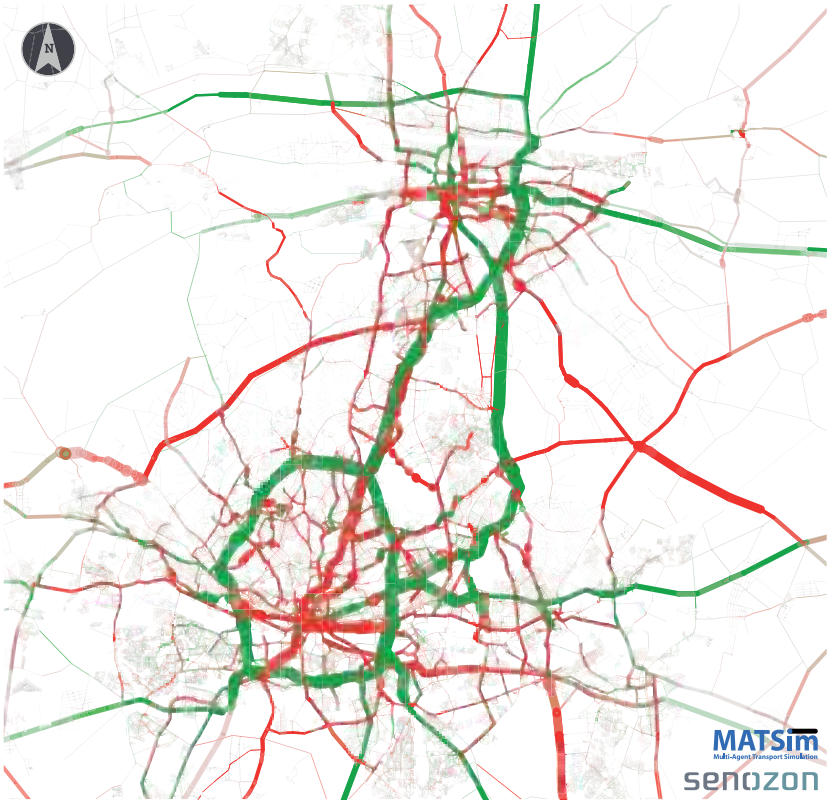


Figure 15.2: Predicted differences in link volumes after introduction of the toll (red: higher volumes, green: lower volumes).

15.4 Invocation

15.4.1 Minimal

A minimum amount of infrastructure is necessary when running roadpricing from the command line. For this, the MATSim JAR, its libraries, *and* the roadpricing JAR need to be downloaded, either from a release or from the nightly builds (Section 44.3.6). After unzipping all zip files, the necessary command is (may need slight refactoring with new formats):

```
java -Xmx2000m -cp MATSim.jar:roadpricing-.../roadpricing-
   ...jar org.matsim.roadpricing.run.RunRoadPricingExample config.xml
```

where `config.xml` needs to contain a section

```
<module name="roadpricing" >
...<param name="tollLinksFile" value="<path>/<tollfilename>" />
</module>
```

The toll file looks like this:

```
<roadpricing type="link" name="abc">
   <links>
      <link id="11">
         <cost start_time="05:00" end_time="10:00" amount="1." />
```

```
            <cost start_time="17:00" end_time="20:00" amount="1." />
        </link>
        <link id="12" />
    </links>

    <!--this is for all links with no cost entry above:-->
    <cost start_time="05:00" end_time="10:00" amount="2.00"/>

</roadpricing>
```

As one can see, there is a section where each link can be entered separately. A separate cost structure for each link is also possible. All links that are listed without a cost structure employ the general cost structure listed at the end. Links not listed are without toll.

15.4.2 Toll Schemes

Link toll The example refers to the "link" toll scheme, indicated by `type="link"`. It charges the amount specified on the link.

Distance toll Another useful scheme is "distance", indicated by `type="distance"`. Here, the `amount` is interpreted as amount per length unit (see Section 2.2.1). This is most useful, with only a list of tolled links and a uniform distance cost for all these links noted at the end of the file.

Area toll The simulation of an *area toll*—i.e., a toll where one has to pay a flat fee for a given time period, often a day, once one drives anywhere inside the area—suffers from a combinatorial challenge: driving through the tolled area early in the day may only pay off if one can re-use the permit later in the day. The code, in principle, addresses that by routing the agent twice: once under the assumption of a zero toll and once under the assumption of a very large toll. Afterward, the toll is added to the generalized cost of the first option, then both options are compared. In the end, the approach suffered from the same consistency burden as the general approach (see end of Section 15.2): the router made the decision about the better variant, rather than leaving the decision to the agent. It should be re-implemented using the same principles as Nagel et al. (2014).

Cordon toll The cordon toll scheme was derived from the area scheme; one could use the same file, listing all area links, for the cordon toll as well. The code ensured that toll was only charged when a vehicle moved from an untolled link to a tolled link—thereby effectively crossing the cordon. One difficulty with this approach: confusion ensues if there is no connected area and several links in sequence are tolled instead. Then, if these links are connected, the toll is only charged on the first of them; if there is a small section missing, perhaps overlooked, the toll is charged again.

SUBPART FOUR

Other Modes Besides Individual Car

CHAPTER 16

Modeling Public Transport with MATSim

Marcel Rieser

16.1 Basic Information

Entry point to documentation:

`http://matsim.org/extensions` → pt

Invoking the module:

The module is invoked by enabling it in the configuration.

Selected publications:

Rieser (2010)

16.2 Introduction

Public transport—or *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

How to cite this book chapter:

Rieser, M. 2016. Modeling Public Transport with MATSim. In: Horni, A, Nagel, K and Axhausen, K W. (eds.) *The Multi-Agent Transport Simulation MATSim*, Pp. 105–110. London: Ubiquity Press. DOI: http://dx.doi.org/10.5334/baw.16.

legs and a simple travel time estimation for modes different than *car*, first case studies resulting in modal share changes were performed using MATSim (Rieser et al., 2009; Grether et al., 2009). This teleportation mode is now available, by default, in MATSim and still a very good fallback option to get a multimodal scenario up and running with as little data as possible.

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

16.3 Data Model and Simulation Features

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

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

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

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

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

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

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

With these data sets, the QSim will simulate all transit vehicle movements. The vehicles will start with their first route stop at the given departure time, allow passengers to enter and then drive along their route, serving stops. At each stop, passengers can enter or leave the vehicle. The simulation generates additional, transit-related events whenever a transit vehicle arrives or departs at a stop, when passengers enter or leave a vehicle, but also when a passenger cannot board a vehicle because

its capacity limit is already reached. This allows for detailed analyses of MATSim's public transport simulations.

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

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

16.4 File formats

16.4.1 `transitVehicles.xml`

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

Public transport vehicle description can be split into two parts; first, 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). Second, actual vehicles must be listed. Each vehicle has an identifier and is a previously specified vehicle type. The following shows an example of a such a file, describing one vehicle and two vehicles of the same type.

```
<?xml version="1.0" encoding="UTF-8"?>
<vehicleDefinitions xmlns="http://www.matsim.org/files/dtd"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="http://www.matsim.org/files/dtd
                           http://www.matsim.org/files/dtd/
                           vehicleDefinitions_v1.0.xsd">
        <vehicleType id="1">
      <description>Small Train</description>
      <capacity>
         <seats persons="50"/>
         <standingRoom persons="30"/>
      </capacity>
      <length meter="50.0"/>
   </vehicleType>
   <vehicle id="tr_1" type="1"/>
   <vehicle id="tr_2" type="1"/>
</vehicleDefinitions>
```

16.4.2 `transitSchedule.xml`

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

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

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

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE transitSchedule SYSTEM "http://www.matsim.org/files/dtd/
                                  transitSchedule_v1.dtd">
<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 and 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 be possible to let a bus run on train links in the simulation.)

The `routeProfile` describes the stops this route serves; 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 an actual departure time. If a bus departs at 7 am, and stop 2 has a `departureOffset` of 3 minutes, this must be read 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 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 return 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 a schedule avoids such issues before the simulation starts, a special validation routine is available:

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

If run, this validator will print out a list of errors or warnings, if any are found, or show a message that the schedule appears to be valid.

16.5 Possible Improvements

While the ability to simulate public transport was a big advance for MATSim, several shortcomings still require attention:

- The data model (and thus, the simulation) does not yet fully support some real world transit lines: for example, circular lines with no defined start and end cannot yet be easily modeled. 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.
- A stop location, as seen by passengers in the real world, is typically modeled as a number of stop facilities in MATSim, detailing different locations where transit vehicles stop (depending on their route and direction). For analysis, one is often interested in aggregated values for such logical stop locations, not for individual stop facilities. Such a logical grouping is still missing in MATSim data format.
- Running simulations with a reduced population sample leads to artifacts when public transport is used. In a simulation with a sampled demand, network capacity is reduced accordingly, to accommodate the fact that fewer private cars are on the road. But because 100 % of public transport vehicles must run (albeit with reduced passenger capacity), calibration becomes difficult. This should be solved, in the future, not by reducing network capacity, but by giving each vehicle and agent a weighting, specifying how much each should count.
- 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 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 already done in the MATSim private car router, will be needed.

- Last, but not least, the current router, based on a modified shortest path algorithm of Dijkstra, can become rather slow and memory-intensive for larger areas with extensive transit offerings. Improved algorithms to generate the routing graph, or different routing algorithms altogether (like the non-graph based Connection Scan Algorithm (Dibbelt et al., 2013)) must be explored in the future.

16.6 Applications

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

- Berlin: the Berlin scenario (see Chapter 53) was one of the first real applications using public transport simulation in MATSim. The road and rail network, as well as the full transit schedule, was converted from a VISUM model. It is still one of the few known models where bus and tram lines share a common network with private car traffic, enabling full interaction between private and public vehicles (like transit vehicles) getting stuck and delayed in traffic jams.
- Switzerland: Senozon AG maintains a model of Switzerland containing the full timetable of all buses, trams, trains, ships, and even cable cars, in the Swiss alps. The schedule data is retrieved from the official timetable, available in a machine-readable format called "HAFAS (HaCon Fahrplan-Auskunfts-System) raw data format".
- Singapore: The model of Singapore (see Chapter 57) makes heavy use of public transport, and continually pushes the boundaries of what is currently possible to simulate. Due to the very large number of buses on Singapore's roads and strong demand for public transport, many extensions had to be implemented to realistically model pt in this context.
- Minibus: The minibus contribution (see Chapter 17) added an optimization layer to public transport functionality in MATSim, allowing automatic generation of an optimized transit schedule for a specific region.
- WagonSim: In the WagonSim contribution (see Chapter 25) public transport simulation was used to simulate rail-bound freight traffic. While the simulation was still moving around transit vehicles and letting passengers enter and leave these vehicles, the scenario had been customized so that vehicles corresponded to freight trains and passengers corresponded to actual goods being transported. Custom implementations of transit driver logic replaced vehicle capacity definition by an alternative definition, ensuring that the trains vehicles represent did not get too long or heavy. The network was constructed so that changing vehicles at stops took minimum time, corresponding to the time needed for switching wagons at freight terminals.

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

CHAPTER 17

The "Minibus" Contribution

Andreas Neumann and Johan W. Joubert

17.1 Basic Information

Entry point to documentation:

`http://matsim.org/extensions` → minibus

Invoking the module:

`http://matsim.org/javadoc` → minibus → `RunMinibus` class

Selected publications:

Neumann (2014)

17.2 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 minibus 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.

How to cite this book chapter:

Neumann, A and Joubert, J W. 2016. The "Minibus" Contribution. In: Horni, A, Nagel, K and Axhausen, K W. (eds.) *The Multi-Agent Transport Simulation MATSim*, Pp. 111–114. London: Ubiquity Press. DOI: http://dx.doi.org/10.5334/baw.17.

The minibus model is integrated in the multimodal multi-agent simulation of MATSim. In the model, competing minibus 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 minibus 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 minibus 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" minibus 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 17.3.

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`.

17.3 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 17.2, 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 BVG (Berliner Verkehrsbetriebe). 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 passenger 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.

CHAPTER 18

Semi-Automatic Tool for Bus Route Map Matching

Sergio Arturo Ordóñez

18.1 Basic Information

Entry point to documentation:

`http://matsim.org/extensions` → gtfs2matsimtransitschedule

Invoking the module:

`http://matsim.org/javadoc` → GTFS2TransitSchedule → `GTFS2MATSimTransitSchedule` class

Selected publications:

Ordóñez Medina and Erath (2011)

Current public transport assignment models adapt network assignment models to work with public transport traffic. Many commercial software products like EMME/2 (Version 2 of EMME), VISUM and OmniTRANS offer sophisticated procedures that include timetable-based route search. However, these models do not include interaction between public transport services and private transport. As mentioned above, the MATSim implementation handles private car traffic and public transport traffic in an integrated way, but it needs accurate public transport line routing on the transport network. While this is usually straightforward for rail-based public transport modes, the routing problem for buses requires more attention; experience shows that assumption of a shortest-path between two consecutive stops leads to unsatisfactory results. To overcome this shortcoming, one can either draw the routes manually or employ map-matching algorithms dependent on tracking data. Due to the burden of manual procedures, and the increasing availability of GPS tracking data, map-matching is becoming increasingly relevant. However, common map matching algorithms are usually not designed to account for the peculiarities of public transport routing; the procedure is very sensitive to errors in network coding, inaccurate bus stop locations and the simplified link shapes in the model.

How to cite this book chapter:

Ordóñez, S A. 2016. Semi-Automatic Tool for Bus Route Map Matching. In: Horni, A, Nagel, K and Axhausen, K W. (eds.) *The Multi-Agent Transport Simulation MATSim*, Pp. 115–122. London: Ubiquity Press. DOI: http://dx.doi.org/10.5334/baw.18.

This section presents a semi-automatic procedure combining public bus routes information (sequences of consecutive stop locations and sequences of geo-referenced points) with a highresolution network (Ordóñez Medina and Erath, 2011). The objective is to obtain a sequence of links for every route of every line and to associate each bus stop with one single link in the network. The procedure was designed to prepare the Singapore scenario public transport extension, but the tools developed can be used to set up any other scenario with similar initial data (timetable and high resolution network).

18.2 Problem Definition

Generally, the problem can be defined as follows. Given:

- a set of stop locations (two-dimensional point coordinates),
- a set of route profiles (sequence of consecutive stops),
- a set of GPS points sequences (sequence of two-dimensional point coordinates), and
- a high resolution navigation network (two-dimensional directed graph with attributes),

the task is to associate each stop with a network link, and translate each route to a network path (connected sequence of links). Figure 18.1 illustrates the problem by providing an example of the available input information and correct output.

Input Information The GTFS (General Transit Feed Specification) is a recent, but already widely-used format for specifying public transport systems, created by Google for feeding its geographic information applications. As of April 2011, the Singapore public transport system featured

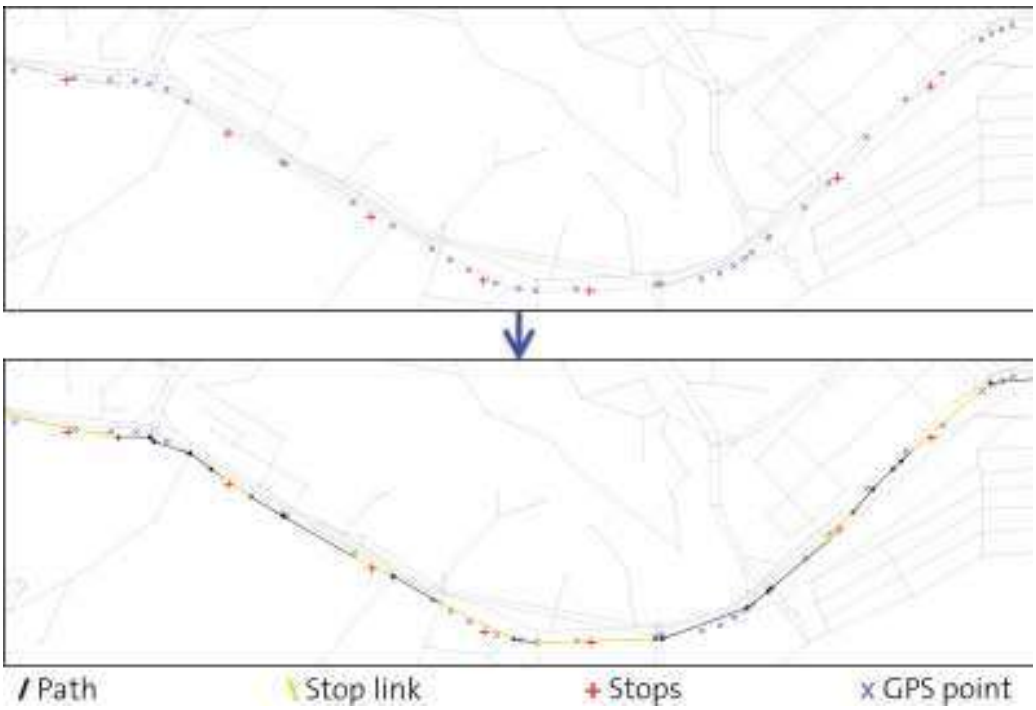


Figure 18.1: Input data and expected solution of the map-matching problem.

Source: Reprinted from Ordóñez Medina and Erath (2011, p.753), Copyright (2011), with permission from Hong Kong Society for Transportation Studies

4 584 bus stops serviced by 355 bus lines, all recorded on GTFS. Each line had several routes, i.e., different outward and return routes (due to one-way streets), as well as different coverage of serviced bus stops on weekdays and weekends. GTFS records the name and location of each bus stop; for bus lines, it records constituent bus routes as a sequence of stops, along with their shape (a sequence of GPS points) as additional information.

The GTFS data must be mapped to a high resolution network; for Singapore, this is a navigation network developed by NAVTEQ. The network is a directed graph where streets and intersections are represented as links and nodes. The links between nodes record attributes like street name, number of lanes, length, flow, free speed and capacity. Nodes are simply recorded as two-dimensional point coordinates. This network has a total of 79 835 links and 43 118 nodes.

Special Restrictions There are some intrinsic characteristics of the public transport system that should be considered serious restrictions. First, when a certain stop is assigned to a network link, this link should be a part of all paths belonging to this stop's routes. In other words: once established, stop-link relationships are fixed for resolving the missing routes. If the GPS points from a route including a specific stop suggest it should be associated with a different nearby link, then all other routes including that stop must be resolved again. Hence, the order in which the routes are resolved is important; it is preferable to resolve those routes first, when we completely trust supporting information quality (e.g., GPS trails).

Second, while many lines run in two directions, with most bus stops having a corresponding stop in the opposite direction (stop located on the other side of the street), this cannot be used to our advantage, because links defined by each return route are different, locations of stops are not necessarily exactly opposite to those in the opposite direction and return routes do not always use the same street.

However, some routes on the same line have an inclusion relationship; in peak hours, segments of bus routes with high demand are served by additional buses running on partial routes to meet demand. In these cases, if a full route is resolved, its partial routes solutions are included.

18.3 Solution Approach

It is not possible to automatically map-match the given GPS position with the network, as standard methods usually require at least 10 points for each link (Schüssler and Axhausen, 2009). In the Singapore GTFS, distance between consecutive points averages about 65 meters, and average link length is about 91 meters; thus, we have fewer than two points per link, on average. Furthermore, not all the routes have GPS points, which inhibits using a full automatic solution; in the Singapore GTFS, there are 38 bus routes without GPS points.

Consequently, the strategy for resolving each route consists of a semi-automatic procedure. Figure 18.2 illustrates the process. First, a simple map-matching algorithm is applied if the route is not part of a bigger route already solved (inclusion relationship described above). In this case, only a previous solution's partition is needed to obtain a first solution. Then, an automatic verification (described below) is performed. If the verification ends with a positive outcome, one can decide to finish the route and save the solution, or to continue editing. If one decides, or is forced, to modify the solution, there are two ways to proceed: changing parameters and running the automatic algorithm again, or editing the solution interactively with a graphical interface editing tool. In both cases, automatic verification must be executed again. If previously saved stop-link relationships are modified, prior routing solutions containing one of the involved stops are erased.

As long as more solutions are obtained, it becomes easier and faster to solve further routes, similar to a machine learning process. This happens for two reasons; first, because of the inclusion relationships that omit the algorithm and second because the increasing number of fixed stop-link relationships relaxes the algorithm (functioning explained in the following section).

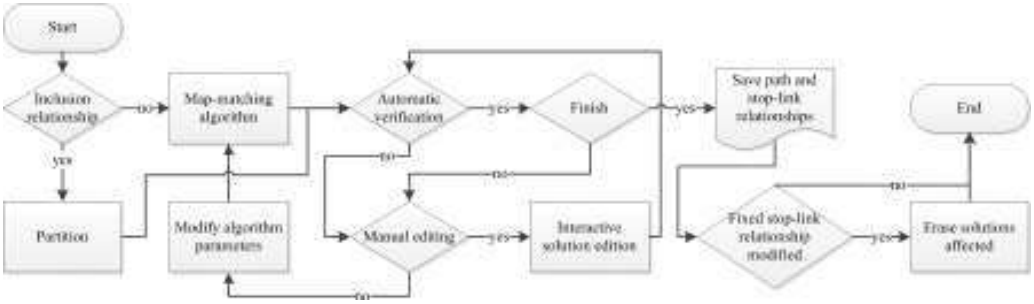


Figure 18.2: Semi-automatic process for one route.

Source: Reprinted from Ordóñez Medina and Erath (2011, p.754), Copyright (2011), with permission from Hong Kong Society for Transportation Studies

18.4 Map-Matching Automatic Algorithm

This algorithm's objective is to generate a solution (path or sequence of connected network links and a set of stop-link relationships) for one route, knowledge of its profile, a sequence of GPS points and a set of stop-link relationships. The algorithm is designed to deal with:

- low GPS point resolution,
- sporadic low network spatial resolution,
- long distances between two express routes stops, and
- understanding that the nearest link to a stop point is not always the correct one.

The route map-matching process is illustrated in Figure 18.3. Except for the first stop, the algorithm solves for each stop in the route profile, a portion of the links sequence (from previous to current) and, if this stop has no fixed link, a set of link candidates pooled from the one link selected.

Link candidates are defined as follows: the *NL* closest links to the stop point, within a distance D_{max}, define a set of candidates. Each set's element could be subjected to more restrictions; the closest point, between the stop point and the infinite line defined by the link, must be inside its line segment and the angle between the link direction and the nearest GPS points sequence direction must be lower than α_{max}.

The link's selection is performed as follows; from the previous stop link to each defined candidate, an A star search algorithm is applied for finding the shortest path. For running this algorithm, each link's cost depends on the link's travel time and distance to the GPS points. A product with flexible exponents was proposed as a first model:

$$C_{link} = \exp \frac{L_{link}}{S_{link}} w_1 \exp D_{GPS} w_2 \tag{18.1}$$

where L_{link} is its length, S_{link} is its free speed, D_{GPS} is its distance to the GPS points sequence and w_1 and w_2 are positive weights with a standard value of 1, but modifiable by the user, according to existence or quality of the GPS points sequence. The definition of D_{GPS} can also be modified; in the simplest approach, it is the minimum distance between the link and all GPS points (point-segment distance). From all calculated paths, the shortest is selected and added to the general route solution. The corresponding link candidate is also related to the stop.

If the current stop has a stop-link relationship, only the shortest path to this stop defines the solution. Thus, the process continues with the next stop in the route profile. If the first stop of the

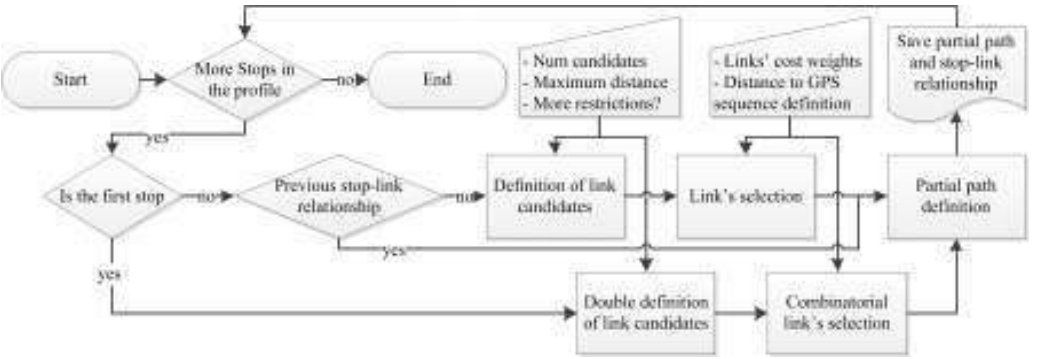


Figure 18.3: Map-matching algorithm.

Source: Reprinted from Ordóñez Medina and Erath (2011, p.755), Copyright (2011), with permission from Hong Kong Society for Transportation Studies

profile has no fixed link, a similar algorithm between the first and the second stop is performed. The definition of candidates' procedure is applied to the first and the second stops. Then, the candidates' selection procedure consists of obtaining the shortest path of all combinations between the two sets of candidates, then selecting the shortest one. This path defines links for both stops.

18.5 Automatic Verification

In this step, accuracy of the routing solution is automatically checked by performing the following ordered verification:

1. Is the path joined?
2. Is the path without U turns?
3. Is the path without repeated links?
4. Does every stop of the route have a stop-link relationship?
5. Is every link related to a stop inside the path?
6. Is the related links' order in the path the same as the corresponding stops' order in the route profile?
7. Is the nearest point between the stop point and the infinite line defined by the link inside its line segment in every stop-link relationship?
8. Are the first and last links of the path related to the first and last stops of the route profile?

Verifications (2), (3) and (7) are not mandatory and can be deactivated through the user interface. User interaction is necessary to (i) cover possible errors, and (ii) include actual route characteristics: some bus routes do include U turns, some repeat exactly the same street, in the same direction, during their travel and the geometric restriction presented in (7) is not always valid in big stop facilities, like bus interchanges.

18.6 Manual Editing Functionalities and Implemented Software

The edit functions' objective is to allow the user to modify the automatically generated routing solution. Even if the automatic algorithm generates a correct solution based on input data,

problems like recent changes in routes, differences in release dates between GPS points and network data, erroneous GPS points, or lack of network element all require manual changes. Although one also could modify and correct the input data, or the generated solution, with direct data modifications, two-dimensional visualization and keyboard-mouse user interaction are two quality attributes that help reduce time and effort. Developed functional requirements and quality attributes are:

1. Visualization: A navigation network is displayed, including all relevant information for working with a single route. This includes the route's profile, given sequence of GPS points, and its current solution (path and stop-link relationships). Selected elements are drawn in a different color. Everything is displayed in a two-dimensional and interactive way, including the cursor location in working coordinates, panning, zoom and view-all options.
2. Selection: Different options for selecting solution elements, or elements from the network, are provided. It is possible to select the nearest link from the solution or from the network, the nearest node from the network, or the nearest stop from the solution, to a point indicated by the user. When a stop that already has a stop-link relationship is displayed, its corresponding link is highlighted as well. If a solution path link is selected and does not have a subsequent link connected, a new one from the network is selected with one click; the selected link is that with the angle most similar to the line defined by the end node of the initial link and a point indicated by the user.
3. Path modification: The first link of the sequence can be added by selecting any network link. If a solution path link does not have a subsequent link connected, it is possible to add one, according to the selection function described in (2). If there are two unconnected sequential links in the solution (a gap), a sub-sequence connecting these links is added, using the shortest path algorithm, with the current parameters. Further, selecting one solution path link, it is possible to delete it, or to delete all links before or after it. Finally, stop-link relationships can be modified by selecting either elements. If the modified relationship was fixed, the user is prevented from modifying the relationship, because the tool will erase the solutions of the routes to which the selected stop belongs.
4. Network modification: New nodes to the road network can be added. In addition, with any node selected, it is possible to add a new link selecting the end node.

These functions were implemented in a software package developed from scratch in Java and using the Java2D library for graphics. The package reproduces the described solution approach, looking for non-solved routes, and running the map-matching algorithm and the automatic verification for each one. Figure 18.4 shows the user interface and a demo video can be accessed at `http://www.vimeo.com/27137889`.

18.7 Conclusion and Outlook

The semi-automatic procedure designed for map-matching bus lines with a high resolution navigation in Singapore was successful, allowing the solving of all bus routes and stops in only ten days, even taking into account the quality of the input information offered, highlighting the low spatial and temporal resolution of the GPS points given for each route. Analysis indicates that reducing manual modification time is the best way to improve the procedure, which can be done by modifying the automatic algorithm to obtain more accurate results for the initial routes to be solved, or in other words, for routes not affected by the learning process.

As GTFS is becoming so popular for defining public transport systems and the code in which this process is implemented is open source, it can be used for matching routes with high

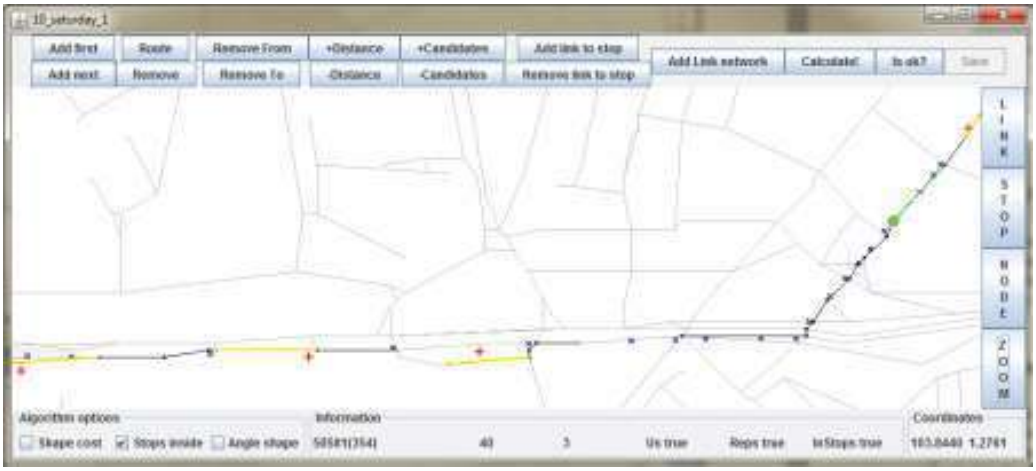


Figure 18.4: User interface of the application to edit automatic solutions.

Source: Reprinted from Ordóñez Medina and Erath (2011, p.757), Copyright (2011), with permission from Hong Kong Society for Transportation Studies

resolution networks of any GTFS-specified place. The tools are available as a MATSim contribution (`GTFS2TransitSchedule`). For generating MATSim simulation scenarios, the procedures have been used by research teams in the province of Gauteng, South Africa, on the Toronto scenario and on a different public transport simulation model developed by SMART-MIT in Singapore.

CHAPTER 19

New Dynamic Events-Based Public Transport Router

Sergio Arturo Ordóñez

19.1 Basic Information

Entry point to documentation:

`http://matsim.org/extensions` → eventsBasedPTRouter

Invoking the module:

`http://matsim.org/javadoc` → eventsBasedPTRouter → `RunControlerWS`, `RunControlerWSV`, `RunControlerWW` classes

Selected publications:

Ordóñez Medina and Erath (2013b)

In public transport route choice, decisions and actions of a particular user depend not only on his/her own preferences, like value of time, crowd avoidance or willingness to pay. They also depend on the decisions and actions of many other public transport users, operators and authorities. Even private transport users' decisions are also involved, as everybody shares the same infrastructure.

This implementation of MATSim used a SBPTR, as mentioned above, meaning that when an agent needed a route for a given start time, origin and destination, the SBPTR found the shortest path in a schedule-based network (assuming public transport vehicles are always on time and always have space). Within the mobility simulation, a vehicle could arrive early or late and/or it could be full, thus not allowing additional passengers to board. With a negative result, the agent obtained a bad score and this plan would have probably been replaced with a more favorable one during the iterative learning process. This scenario's problem occurred when the agent tried to

How to cite this book chapter:

Ordóñez, S A. 2016. New Dynamic Events-Based Public Transport Router. In: Horni, A, Nagel, K and Axhausen, K W. (eds.) *The Multi-Agent Transport Simulation MATSim*, Pp. 123–132. London: Ubiquity Press. DOI: http://dx.doi.org/10.5334/baw.19.

find a new route for the same start time, origin and destination, the public transport scheduled network shortest path remained the same; agents could not improve their experiences by changing the route.

To address this shortcoming, a new EBPTR (Events-Based Public Transport Router) was proposed (Ordóñez Medina and Erath, 2013b), modeled, implemented and tested. It took the given schedule as a base for the first iteration, but updated information on travel times, occupancy of the public transport vehicles, and waiting times was propagated between subsequent iterations. Thus, when same day executions were performed, new routes could be generated for the same start time, origin and destination, because the system is remembered delayed bus services (longer travel times), or train services where the vehicle arrived full (longer waiting times). However, the network used to route agents required a new topology to account for such variables. This approach allowed then to account for emergent phenomena; in situations where overcrowded vehicles prohibited boarding, it made sense for some agents to travel a few stops in the outbound direction. They could then transfer to an inbound vehicle with sufficient capacity and board. Although more memory was needed, similar or even better computation times were achieved when shortest path calculations awe performed, due to the simpler network topology. Furthermore, to achieve user equilibrium required a significantly smaller number of iterations.

19.2 Events-Based Public Transport Router

A new EBPTR was developed for MATSim to more realistically model public transport route choice, where agents learn, over time, that transit vehicles are not always on time, do not always have sufficient space to allow boarding and trips with more comfort are often preferable.

Network Topology Figure 19.1(b) shows the structure of the proposed public transport network, compared with the original structure (Figure 19.1(a)). Inspired by the network designed by Spiess and Florian (1989) this implementation had two types of nodes. The first type represented a stop facility (green-black squares) as point in space, while the second type (yellow-red dots) represented a stop-route relation which could be seen as a physical or virtual platform for each line passing a particular stop facility. For example, different platforms in a metro system needed to be modeled as different stop facilities, because different services arrived at each platform and walking paths were needed to change from one platform to another. For bus stop facilities, they represented virtual platforms; in reality, buses from different lines serving the same bus stop would normally use the same physical infrastructure e.g., a bus bay. To connect those nodes, there were four types of links. The in-vehicle links joined two consecutive stop-route nodes in the direction of the correspondent route. The boarding links connected a stop node with each corresponding stop-route node. The alighting links were opposite, connecting stop-route nodes with their corresponding stop node. Finally, walking links connected a stop node with all other stop nodes located within walking distance.

Link Costs Each link in this network had a related time-dependent disutility function. Different costs were saved for different times in the day for a given time bin (at this time, 15 minutes). In-vehicle link disutilities depend on vehicle travel time, travel distance, level of occupancy and a fare rate, if this system is distance-based. Boarding link disutilities depended on waiting times, a transfer cost, and a fixed fare if this system was entry-based; thus it was possible to relate specific stop-route waiting times to these links. As the first waiting link was not a transfer, this cost had to be subtracted from the whole path cost, but this detail did not affect the shortest path calculation.

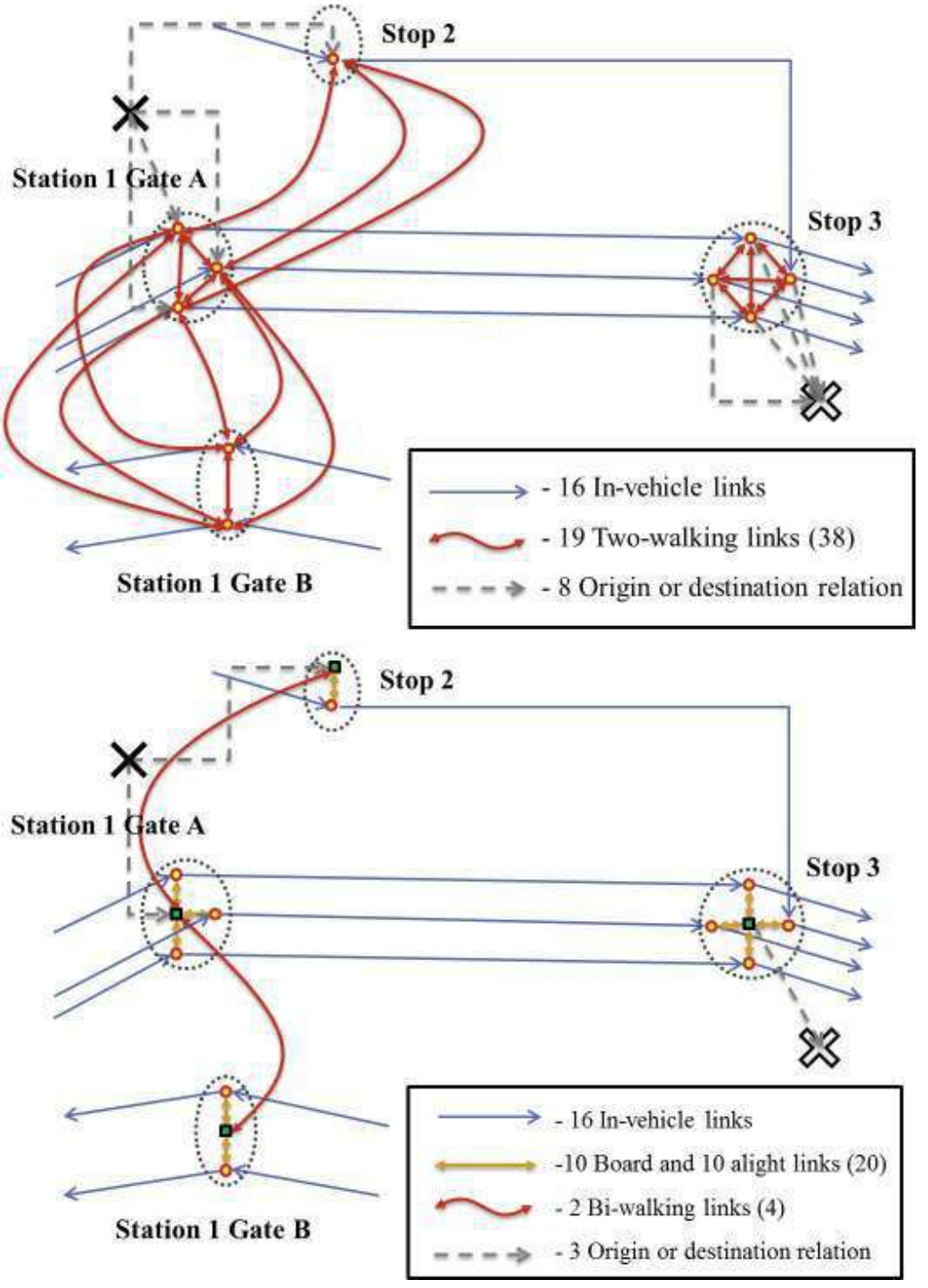


Figure 19.1: Comparison of the network topologies of the schedule-based transit router (a) and the new events-based transit router (b).

Alighting links had no associated cost, but a fare could be related to them. Finally, walking links depended on the walking travel time and distance. Equation (19.1) shows linear versions of these functions used in this model assuming a distance-based fare system.

$$\begin{aligned} C_{iv}(t) &= (\beta_{iv} * t_{iv}(t))(1 + g(p_{oc}(t))) + \beta_{vd} * l_{iv} + f_{iv} * l_{iv} \\ C_{bo}(t) &= \beta_{wt} * t_{wt}(t)) + c_{tr} \\ C_{al}(t) &= 0 \\ C_{wk}(t) &= \beta_{wk} * t_{wk} + \beta_{wd} * l_{wk} \\ C_{path}(t) &= \sum C_{iv}(t') + \sum C_{bo}(t') + \sum C_{al}(t') + \sum C_{tr}(t') - c_{tr} \end{aligned} \tag{19.1}$$

C_{path}: Total cost of the path.
C_{iv}: Cost of one in-vehicle link.
C_{bo}: Cost of one boarding link.
C_{al}: Cost of one alighting link.
C_{wk}: Cost of one walking link.
β_{iv}: Personalized cost per unit of time traveling in a vehicle.
β_{vd}: Personalized cost per unit of distance traveling in a vehicle.
β_{wt}: Personalized cost per unit of time waiting in a stop.
β_{wk}: Personalized cost per unit of time walking.
β_{wd}: Personalized cost per unit of distance walking.
c_{tr}: Personalized cost for making a transfer.
f_{iv}: Vehicle dependent fare rate by distance traveled.
$t_{iv}(t)$: In-vehicle travel time (from Stop-stop travel times structure).
$t_{wt}(t)$: Waiting time (from Stop-route waiting times structure).
t_{wk}: Walking time.
l_{iv}: In-vehicle distance.
l_{wk}: Walking distance.
$p_{oc}(t)$: Occupancy level in the in-vehicle link (from Route-stop occupancy structure).
$g(p)$: Simplified function of how occupancy level increases the cost (Equation (19.2)).

$$g(p) = \begin{cases} 0 & if \quad p \leq p_{sit} \\ r_{sta} * p + b_{sta} & if \quad p_{sit} < p < 1 \\ b_{full} & if \quad p = 1 \end{cases} \tag{19.2}$$

p_{sit}: Occupancy level when no more seats are available.
r_{sta}, b_{sta}: Parameters of percentage increase in discomfort from standing in the vehicle.
b_{full}: Maximum percentage increase when the vehicle is full.

Shortest Path Algorithm To find a public transport route between an origin and a destination, for a given time of day, the applied method was the same as currently implemented in MATSim; first, the algorithm looked for the stop-nodes within walking distance from both origin and destination. An initial cost was associated with each of these stop-nodes, according to access and egress walking times. Then, starting from all the origin-stop-nodes with a given access cost, a multi-node time dependent Dijkstra algorithm found the shortest path, to the destination-stop-nodes with related egress costs. Thus, the path determined the best O-D (Origin-Destination) combination as well. The algorithm was time-dependent because it recognized that while it proceeded through the path, time advanced; thus, different costs are obtained from the links while time advanced. The total disutility of this path was compared with the cost of a full walking trip. If the cost is less, the path is converted to a sequence of stages: in-vehicle stages for each in-vehicle link in the path and walking stages for each walking link. Boarding and alighting links were ignored for this conversion.

Structures to Save Travel Times, Waiting Times and Vehicle Occupancy As mentioned earlier, the mobsim of MATSim generated atomic units of information called events, which described changes for each person, e.g., boarding or alighting: each vehicle, e.g., entering and leaving a link during the simulation. The goal was to save information on public transport experience in one simulation and find better public transport routes for agents in the next iteration. This feedback mechanism was already implemented in MATSim for private transport; the car router used each saved link's time-dependent travel times from a previous iteration to calculate better routes in the road network, by changing the costs of the links. To allow the EBPTR to learn from the previous iteration, information about (a) stop-stop travel times, (b) stop-route waiting times and (c) route-stop-stop vehicle occupancy, was required.

- Stop-stop travel times: To account for public transport vehicle delays, travel time between consecutive stops had be saved. Two stops are consecutive if they were consecutive for at least one public transport route. A first option was using the previously discussed travel times structure that saved time-dependent travel times for each road network link. Because a vehicle had to follow known road links between two consecutive stops, these travel times could be summed. One problem: this structure accounted for all the vehicles in the network, but travel times of cars and buses were very different, particularly in links with public transport stops. Thus, a special structure was implemented to save these stop-stop travel times. The structure averaged all the public transport vehicle times from one stop to the next during a certain time bin. More specifically, each value comprised the time from when the vehicle arrived at a certain stop until it arrived at the next stop, denoted in the simulation by consecutive `VehicleArrivesAtFacility` events. This meant that the first stop waiting time and all queue times (if the vehicle had to queue before the bay or platform was available) were included. In other words, when an agent routed the first in-vehicle link of each trip, the full dwell time would be included. Hence, this agent assumed it was the first passenger entering the vehicle. For all the other in-vehicle links the in-vehicle waiting was included. These stop-stop times were the main component of the in-vehicle link disutilities.
- Stop-route waiting times: Waiting times are a fundamental aspect of public transport route choice and can be long due to vehicle delays (i.e., due to the stop location), or full public transport vehicles of one or several consecutive services (i.e., due to the route demand and stop position within the route). For that reason, waiting times were saved for each stop-route relation. Similarly, the structure averaged all agent waiting times in a certain stop, for a certain route, during a certain time bin in the day. More specifically, each value comprised the time from when the agent arrived at the public transport stop until it entered the vehicle, denoted in the simulation by consecutive `AgentArrivesToFacility` and `PersonEnterVehicle` events. These waiting times were the principal component of boarding link disutilities. If no observations were found for a certain stop-route-time, the model returned half the corresponding headway, specified by the transit schedule.
- Route-stop occupancy: By accounting for occupancy level, one can model routing decisions where people take longer/slower routes to feel more comfortable in emptier vehicles, i.e., valuing a higher chance to travel while seated. Occupancy depends on specific route demand and the stop position within the route. Here, occupancy was assumed to be constant between two consecutive stops. When a vehicle departed from a certain stop (denoted in the simulation as `VehicleDepartsFromFacility` event) this structure averaged the occupancy level with the other vehicles on the same route departing from the same stop during the same time bin. As there were only a few vehicles recorded for each time bin, it was unlikely to find observations for a specific bin. In this case, the structure returned the value of the next time bin, where at least one observation was found for the corresponding stop and route.

19.3 Functional Results

Relaxation Process The number of iterations needed by MATSim's co-evolutionary algorithm to reach a stable state was a critical variable; efforts were made to reduce it (Meister et al., 2006; Fourie et al., 2013).

The EBPTR effectively reduced the iterations public transport users needed to reach equilibrium. Using a 25 % sample of the Singapore scenario, Figure 19.2 shows average score plan evolution for 355 207 agents over 100 iterations. These 100 iterations were executed four times to use both routers for two different replanning strategies. Agents saved five plans in memory. At iteration 0, both EBPTR and SBPTR started with routes described in the schedule; however, the EBPTR returned routes that performed better in this first simulation. This occurred because, for each pair of consecutive stops, the EBPTR used the average of all scheduled route times that contained this pair as the first estimate. On the other hand, the SBPTR used the specific scheduled time of the corresponding route. Results indicated the average stop-stop time seemed to be a more reliable estimate for this first iteration.

For the rest of the iterations, the Figure 19.2 shows how the scores evolved. The first replanning strategy stipulated that 30 % of the agents were re-routed at each iteration. This evolution is shown in the first graph of the figure. Using SBPTR, agents received the same route over and over again as the start time, origin and destination did not change between iterations. Small variations in scores occurred because of the stochastic simulation nature explained above. Although scores started in the same range, using EBPTR allowed better-performing routes to be found within a very small number of iterations.

For a more realistic comparison, a second replanning strategy was tested, where just 20 % of the agents were re-routed and the activity start times were modified randomly within a half an hour for 10 % of the agents. The second graph of the figure shows how both routers managed to improve agents' plan scores. But with the EBPTR, number of iterations needed to achieve the average executed score, achieved after 100 iterations for the SBPTR (120), was only 5. The target marginal score, as a measure of change in score over iterations, was taken arbitrarily as 0.1 utilities per iteration, or the rate produced after 200 iterations with the SBPTR. In contrast, this target rate was achieved after 77 iterations with the EBPTR, a 2.6 improvement factor .

Modeling Advantages Because of the links disutility function in the proposed network account for aspects like waiting times or occupancy levels and because MATSim allows for modeling heterogeneity among agents, the router could be a very powerful tool to model observed emergent behavior in public transport route choice. In Singapore, like many other crowded cities in the world, some commuters decide to travel backwards for a few stops and then transfer to a train in the opposite direction to find a seat or space in a public transport vehicle Chakirov and Erath (2011). With the SBPTR this kind of least cost path could not be found, but with the newer proposal, this was possible. Although proportions did not match actual observations as the Singapore scenario lacked appropriate and calibrated utility parameters for traveling and waiting time under crowded conditions, Figure 19.3 shows totals of people traveling backwards from different stops in the island after 100 iterations (see Figure 19.2 (a)).

19.3.1 Comparing Quality Attributes With the Current Implementation

Computation Time The tests described next were executed using 12 computational nodes, accessing 70 GB of shared memory, using the Singapore scenario described in Chapter 57. Before the first iteration, if plans were not routed, MATSim prepared every agent with an initial route. As mentioned before, the stop-stop travel times and stop-route waiting times were initially taken from the schedule. Because of its simpler network structure the EBPTR took 01:17:35 to initially

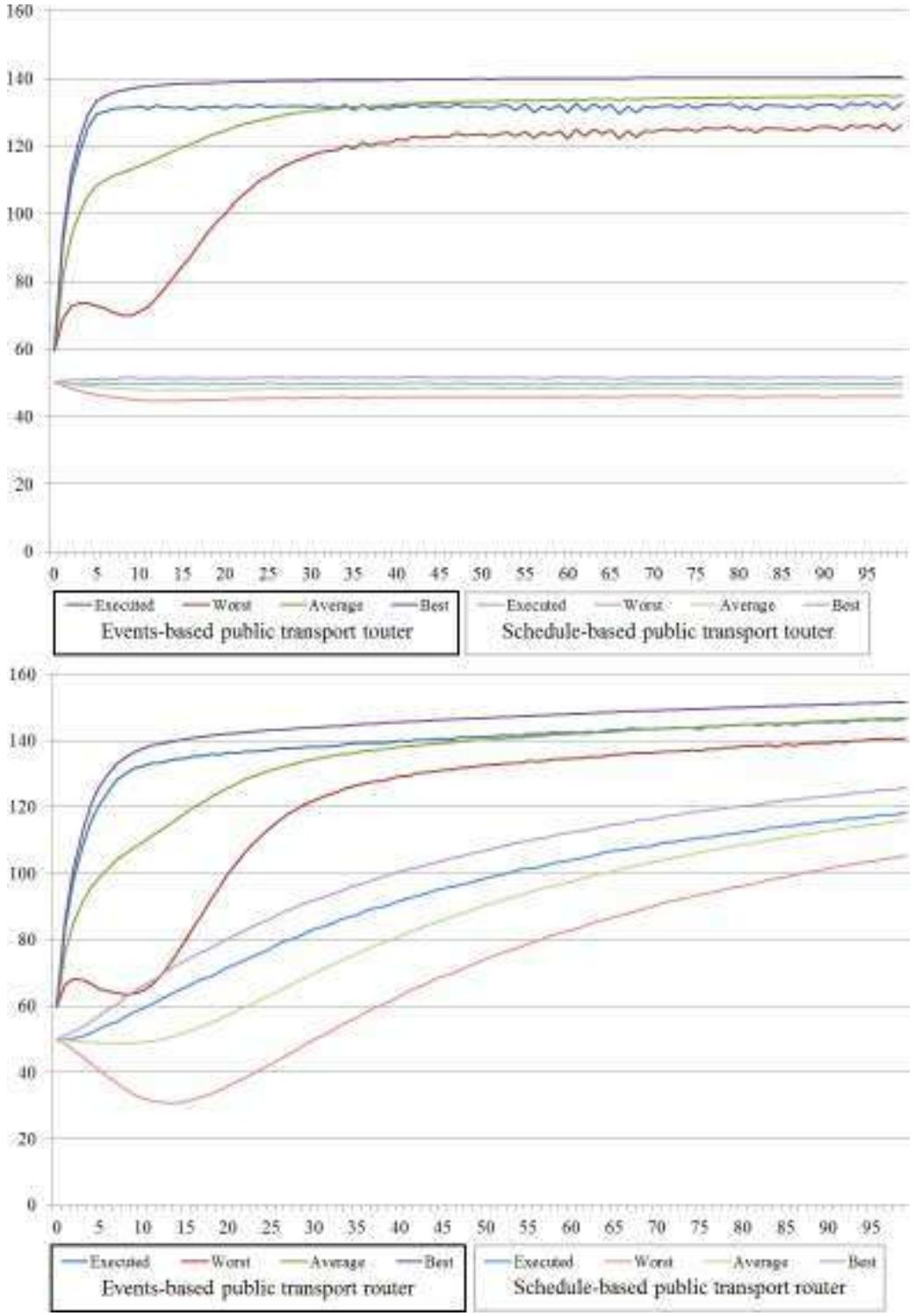


Figure 19.2: Comparison of score evolution: a) 30 % re-route, b) 20 % re-route and 10 % time allocation.

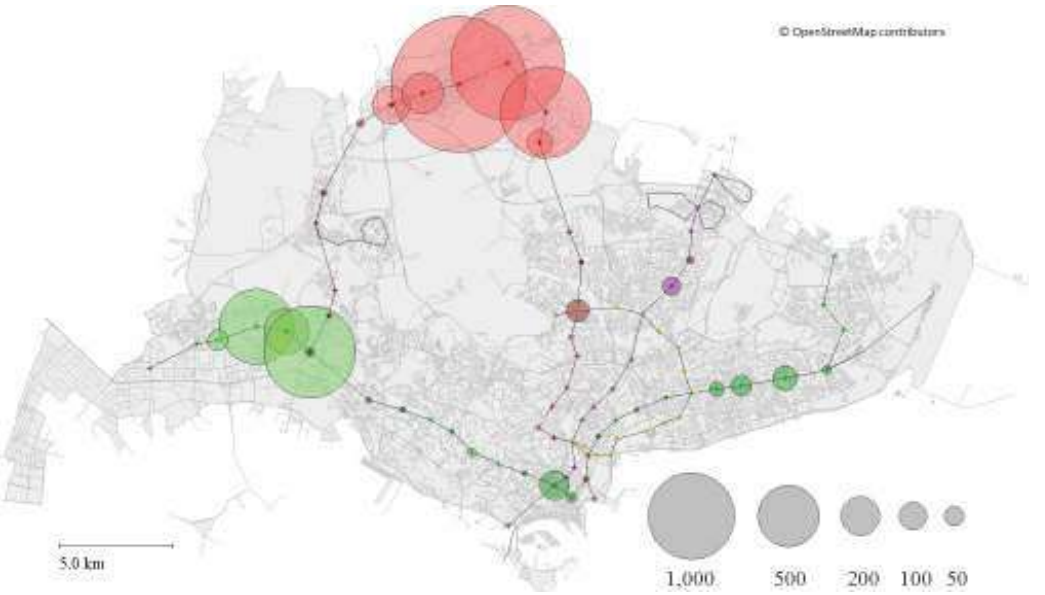


Figure 19.3: Number of agents traveling backwards at each MRT (Mass Rapid Transit, Singapore) station of the Singaporean rail system.

route the 355 207 users, compared with 01:28:55 needed by the SBPTR, producing a performance gain of about 12.7 % for this scenario. When running MATSim iterations with the EBPTR, computation times principally changed in two processes: mobility simulation (mobsim) and replanning. Figure 19.4 shows computation times measured for the first 20 iterations of the process. Although the EBPTR needed more time in mobsim, it continued to require considerably less time for rerouting during the replanning, due to a simpler network topology. The longer mobsim time was due to information saving in the new structures during the simulation. However, on average, the EBPTR outperformed SBPTR, per iteration, by about 3 minutes or 11 %. As mentioned above, 2.6 times more iterations were needed for the SBPTR to achieve a specific point in the relaxation process. For 77 iterations with the EBPTR, computation amounts 35:25:43, and for 200 iterations with the SBPTR, computation amounts 99:10:51; a 2.8 improvement factor in our experimental setting.

Memory Consumption The EBPTR needed more memory than the SBPTR, because the EBPTR managed more information. The necessary extra memory was allocated to the three structures described before. Given the Singapore scenario conditions described, the extra memory was calculated as follows. One numeric value needed eight Bytes, and with a time bin of 15 minutes, 120 bins were needed for 30 hours. The Stop-stop travel times structure saved two values (average and number of observations) for each time bin and each pair of consecutive stops. The number of pairs for the Singaporean public transport system was 6 602. Thus, this structure needed approximately 12.7 MB. Similarly, the stop-route waiting time structure saved two values (average and number of observations) for each time bin and each pair of stop/route combinations. The number of stop/route relations for the Singaporean public transport system was 27 156. Thus, this structure needed approximately 52.1 MB. Finally, the vehicle occupancy structure saved the average and number of vehicle occupancy observations for 26 353 route-stop relations for each of the 120 time bins, requiring approximately 50.7 MB. In total, less than 120 MB were needed for the three structures.

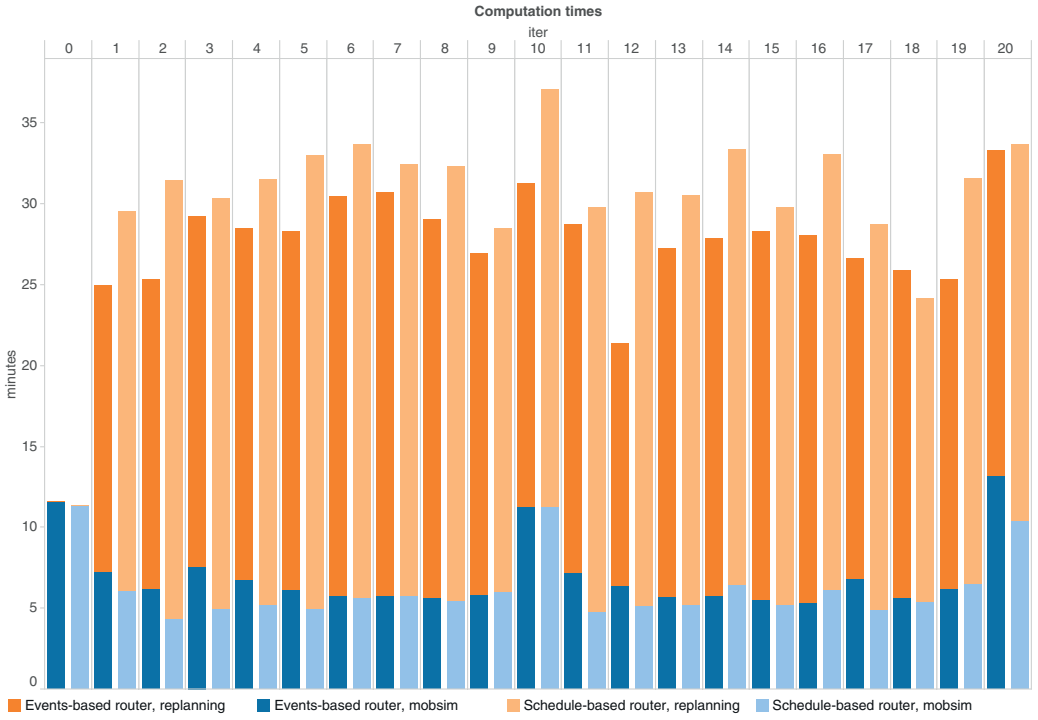


Figure 19.4: Comparison of computation times for 20 iterations.

On the other hand, the size of the network where public transport routes were calculated was smaller for the EBPTR. Although, in the case of Singapore, it created 31 939 nodes compared with 27 156 of the SBPTR (4 783 new stop-nodes), the number of links is dramatically smaller. The SBPTR created 424 070 walking links and 26 353 travel links (450 423 in total). The EBPTR created the same 26 353 travel links, plus 27 156 boarding links, plus 27 156 alighting links and just 4 390 walking links (85 055 links in total); less than a fifth in total. As a node needed 48 bytes and a link 128 bytes, the SBPTR needed roughly 46.8 MB more memory for links and just 229.6 KB less for nodes. The EBPTR saved 46.5 MB for the network, concluding that in total the SBPTR needed 70 MB less memory. This quantity was negligible compared with the total memory needed for the whole simulation (more than 40 GB).

19.4 Conclusion and Future Work

In this work, a new public transport router for MATSim was designed, implemented and tested. It produced more diverse routes in large scale scenarios, taking into account many complexities of urban public transport systems. On the supply side, the system simulated congestion, public transport vehicles occupancy levels, queues in public transport stops, bay sizes, and bus or train bunching. On the demand side, in addition to commonly used factors like in-vehicle time, number of transfers and walking time, the new router took disutility of additional waiting time due to congestion or overcrowded vehicles, comfort level inside public transport vehicles and preference heterogeneity among agents for all mentioned factors into account.

The utility of the new approach was tested in a large scale Singapore scenario. Using a simplified public transport only simulation, 100 iterations of a 25 % scenario (355 207 agents) with 30 % of the agents re-routing each iteration took just 45 hours approximately, or about 27 minutes per

iteration, using 12 cores and 70 GB of memory. The computation decreased by 11 %, compared to the standard MATSim. If just 20 % of the plans were re-routed, using 35 cores accessing 85 GB of memory, the time per iteration would be reduced to less than 13 minutes, achieving 100 iterations in less than one day. But, most importantly, for computation time gains, we showed that the proposed events-based router was able to reach a steady state in a much smaller number of iterations.

If the proposed router works better than the original one, should it be changed? The current scheduled-based router of MATSim would still be relevant if the topology of its network were changed for the proposed one. It should also be applied to scenarios where the public transport system operates very reliably and punctually, with few cases of overcrowding. In that case, routing calculations would be as fast as the events-based router (with the new network structure), and the mobility simulation would be faster, as no information (in-vehicle time, wait time and occupancy) would be needed. In other words, it could be applied to city models where public transport users can reliably plan their trips using only a timetable.

Scrutinizing the resulting network loading, the biggest potential advantage of the proposed events-based router was its capacity to generate emergent behavior in congested public transport systems, in line with actual observations. Future research should aim at estimating the various route choice behavior parameters corresponding to the functionalities of the proposed system and calibrating the simulation. Although the values used came from a stated preference survey commissioned by the Land Transport Authority for the case of Singapore, advanced studies could, for example, be tailored to quantify preference heterogeneity. Furthermore, results from work in progress about the value of a seat in Singapore and discomfort disutility can improve prediction confidence. Finally, information from the Singapore smart card data could be used for revealed preference estimation of further behavioral parameters, like quality of a transfer described, e.g., by the number of escalators, to further refine the system.