

CHAPTER 20

Matrix-Based pt router

Kai Nagel

20.1 Basic Information

Entry point to documentation:

<http://matsim.org/extensions> → matrixbasedptrouter

Invoking the module:

<http://matsim.org/javadoc> → matrixbasedptrouter → RunMatrixBasedPTRouterExample class

Selected publications:

Section 3.1 of Nicolai and Nagel (2015); Röder et al. (2013)

20.2 Summary

The matrix based PT (Public Transport) router reads a list of PT stops, and constructs “teleported” PT routes using the stops nearest to origin and destination. That is, each resulting trip will approximately look as follows:

```
<act type="previous" ... />
<!-- begin trip -->
<leg mode="walk" ... />
<act type="ptInteraction" ... />
<leg mode="pt" ... />
<act type="ptInteraction" ... />
<leg mode="walk" ... />
<!-- end trip -->
<act type="next" ... />
```

How to cite this book chapter:

Nagel, K. 2016. Matrix-Based pt router. In: Horni, A, Nagel, K and Axhausen, K W. (eds.) *The Multi-Agent Transport Simulation MATSim*, Pp. 133–134. London: Ubiquity Press. DOI: <http://dx.doi.org/10.5334/baw.20>. License: CC-BY 4.0

The attributes of the walk and the PT legs will be computed from the coordinates of the locations in the same way as teleportation routing (see Section 4.6.2.2), and then taken at face value in the mobsim (see Section 4.6.1.2).

Travel times and travel distances between PT stops can alternatively be given by corresponding matrices. This is particularly useful if a PT assignment exists and such information can be extracted from that. This was used by Röder et al. (2013) and by Zöllig Renner (2014).

CHAPTER 21

The “Multi-Modal” Contribution

Christoph Dobler and Gregor Lämmel

21.1 Basic Information

Entry point to documentation:

<http://matsim.org/extensions> → multimodal

Invoking the module:

<http://matsim.org/javadoc> → multimodal → `RunMultimodalExample` class

Selected publications:

Dobler and Lämmel (2014)

21.2 Introduction

MATSim’s standard mobsim, QSim, has recently been enabled to model multimodal scenarios as shown in Section 4.6.

In this chapter,¹ an earlier approach to handle multimodal scenarios, the multimodal link contribution, is presented. As shown below, it is a very efficient approach, that considers persons’ biking and walking speeds to improve the teleportation estimates for these modes, whereas mode interactions are not taken into account.

¹ Parts of this chapter are based on work published at the 6th International Conference on Pedestrian and Evacuation Dynamics in Zürich Dobler and Lämmel (2014).

How to cite this book chapter:

Dobler, C and Lämmel, G. 2016. The “Multi-Modal” Contribution. In: Horni, A, Nagel, K and Axhausen, K W. (eds.) *The Multi-Agent Transport Simulation MATSim*, Pp. 135–140. London: Ubiquity Press. DOI: <http://dx.doi.org/10.5334/baw.21>. License: CC-BY 4.0

21.3 Modeling Approach and Implementation

21.3.1 Multi-modal Link Contribution

Figure 21.1 shows the implementation's basic concept—a multimodal contribution is added to each link object in the mobsim.

While traffic flow dynamics are simulated by MATSim's mobsim using a queue model, these flows are not taken into account in the multimodal contribution. Examining typical pedestrian and cyclist traffic flows shows that congestion is very rare compared to vehicular traffic, justifying application of this simplistic approach over a scenario. For regions with higher traffic flows, this simple model loses accuracy, but still outperforms the teleportation approach, which MATSim uses by default.

Each multimodal link contribution uses a priority queue to manage all agents traveling on that link using a non-motorized mode. The queue orders the agents based on their scheduled link leave time (see Figure 21.2). This time is calculated when an agent enters a link and is based on parameters like the agent's age and gender, as well as the links' steepness. In each time step, it is checked whether the queue contains agents who have reached their link leave time and thus must be moved to their route's next link. An agent's position on a link is not determined by the model. However, under the assumption that agents move with constant speed, their position can be interpolated. This approach is computationally very efficient, because computation effort is created only when an agent enters or leaves a link but not when it is traveling along a link. Additionally, agents can travel at different speeds, so can overtake each other.

21.3.2 Travel Times

Walk travel time calculation is based on results of a comprehensive literature review by Weidmann (1992). Starting point is a normally distributed reference speed of 1.34 meters per second with a standard deviation of 0.26 meters per second, which leads to an individual reference speed for each person. FGSV (2009) and Transportation Research Board (2010) report comparable,

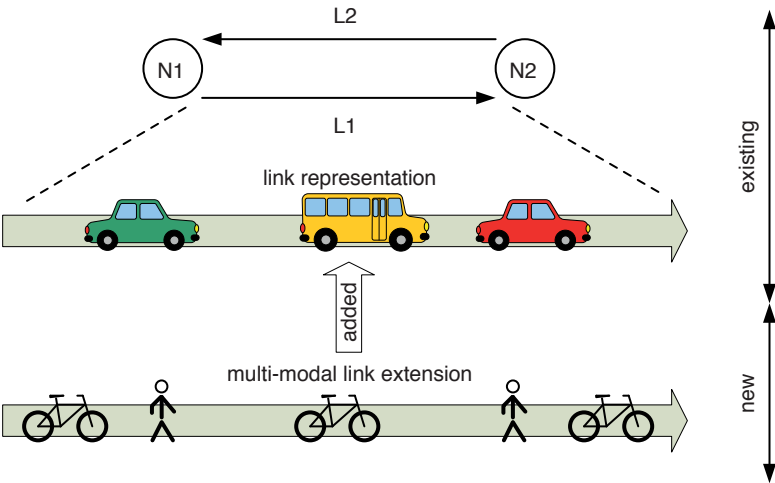


Figure 21.1: Multi-modal link contribution.

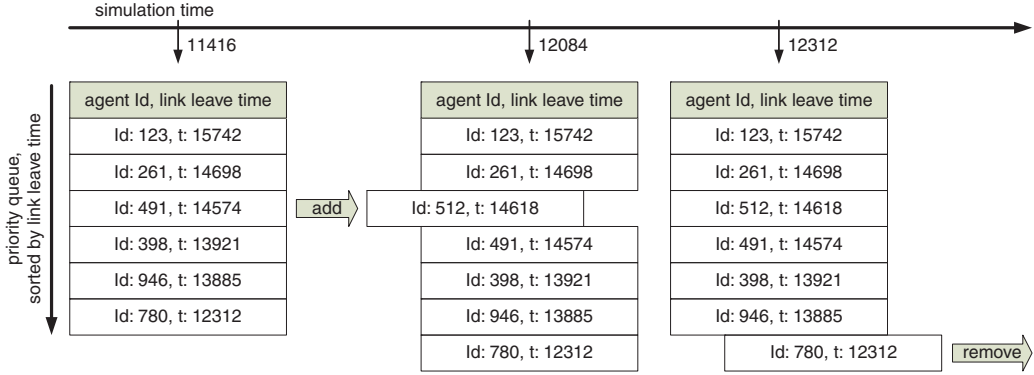


Figure 21.2: Link representation in the simple model.

At time 12 084 seconds from midnight, agent 512 enters the link and is—based on its calculated link leave time 14 618 seconds from midnight—inserted into the queue. At time 12 312 seconds from midnight, agent 780 has reached its leave time and is then removed from the queue.

but less detailed data. If a trip’s purpose is known, a person’s reference value can be adjusted (commuting 1.49 meters per second, shopping 1.16 meters per second, leisure 1.10 meters per second; see FGSV, 2009). Using the reference speed and referencing a person’s age, gender and statistical spreading, a personalized speed is calculated (see Figure 21.3(a)). Finally, to calculate the person’s travel time on a specific link, influence of the link’s steepness on the person’s speed is taken into account (see Figure 21.3(b)). The combination of person-specific attributes and link steepness is shown in Figure 21.3(c).

As a result, a person’s speed on plain terrain is calculated as:

$$f_{\text{person}} = f_{\text{statistical spreading}} \cdot f_{\text{gender}} \cdot f_{\text{age}} \quad (21.1)$$

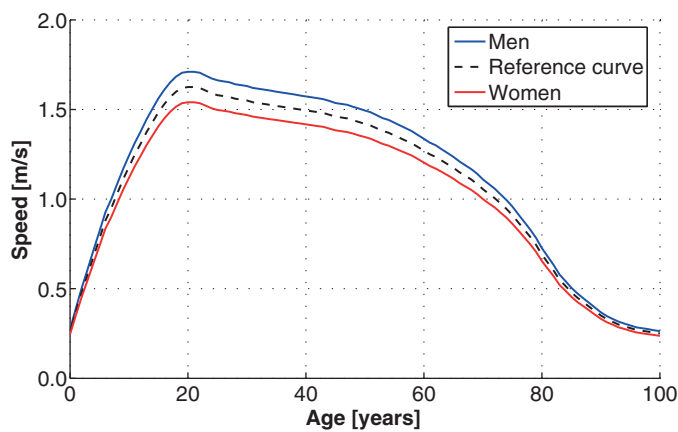
$$v_{\text{person, walk}} = v_{\text{reference, walk}} \cdot f_{\text{person}} \quad (21.2)$$

A link’s steepness is incorporated as:

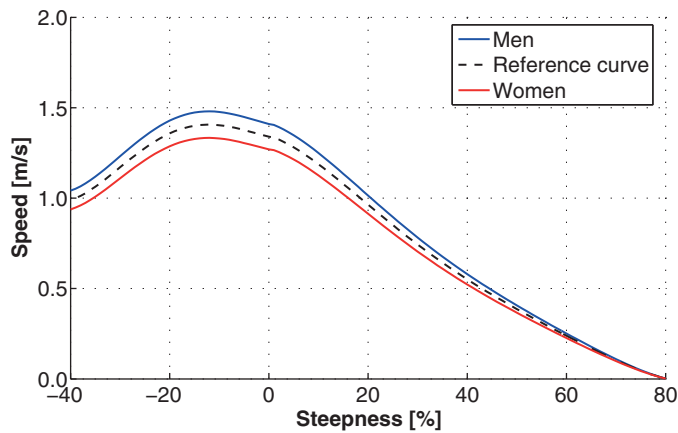
$$v_{\text{person walks on link}} = v_{\text{person, walk}} \cdot f_{\text{steepness}} \quad (21.3)$$

The speed of cyclists is determined using results from Parkin and Rotheram (2010). Starting point is, again, an individual’s speed based on a normal distributed ($\mathcal{N}(6.01, 1.17)$) reference speed. Once more, a person’s speed is calculated by accounting for age and gender (see Figure 21.4(a)).

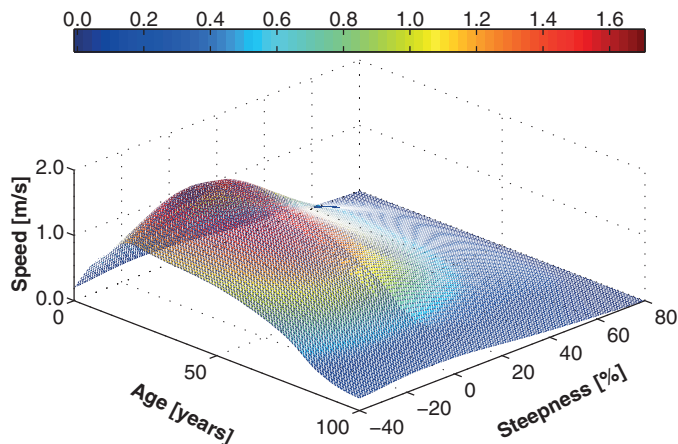
When calculating the steepness factor, one must define whether a link goes uphill or downhill. When going uphill, the person’s speed is reduced by a factor based on the grade and a reference factor of 0.4002 meters per second, which is scaled by the same factor as the person’s reference speed. i.e., the speed drop of slow people is lower than the drop of fast people. When bike speed drops below walk speed, which happens at a grade of approximately 12 %, it is assumed that the person switches to walking (see Equation (21.5)). For downhill links, a reference factor of



(a) Age dependent speed.

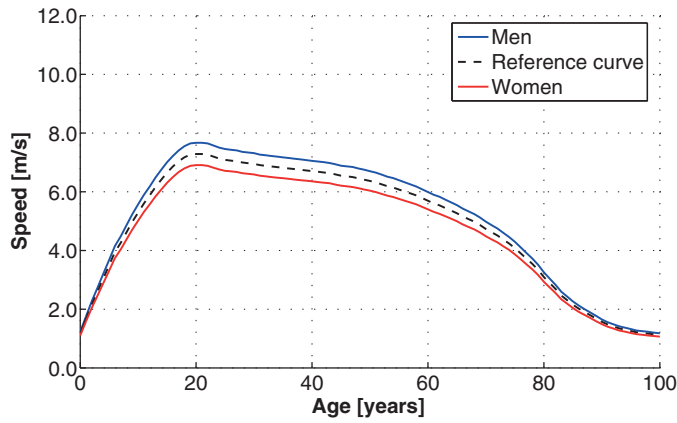


(b) Steepness dependent speed.

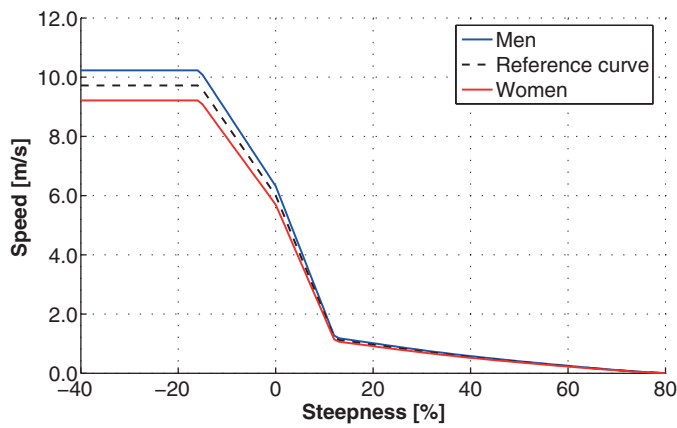


(c) Age and steepness dependent speed.

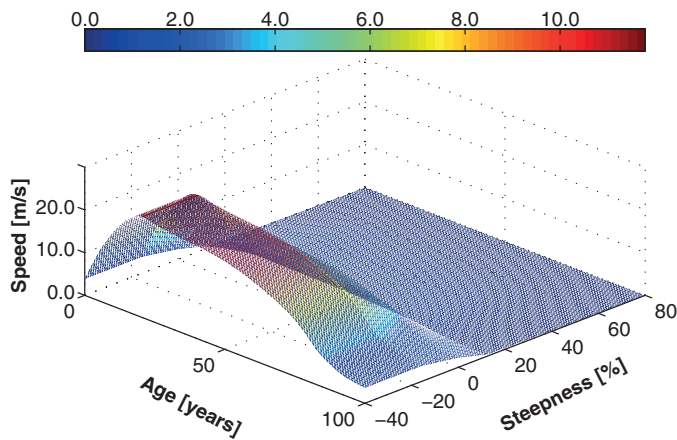
Figure 21.3: Age and steepness dependent speed of pedestrians.



(a) Age dependent speed.



(b) Steepness dependent speed.



(c) Age and steepness dependent speed.

Figure 21.4: Age and steepness dependent speed of cyclists.

0.2379 m/s is used. Additionally, it is assumed that cyclists limit their speed to 35 kilometers per hour (9.7222 meters per second; see Equation (21.6)).

$$v_{\text{person, bike}} = v_{\text{reference, bike}} \cdot f_{\text{person}} \quad (21.4)$$

$$v_{\text{person, uphill}} = \max \begin{cases} v_{\text{person, bike, flat}} - 0.4002 \cdot |\text{grade}| \cdot f_{\text{person}} \\ v_{\text{person, walk, uphill}} \end{cases} \quad (21.5)$$

$$v_{\text{person, downhill}} = \min \begin{cases} v_{\text{person, bike, flat}} + 0.2379 \cdot |\text{grade}| \cdot f_{\text{person}} \\ 9.7222 \end{cases} \quad (21.6)$$

Another parameter affecting pedestrian and cyclist speed is the crowd density of the link where they are physically present. Data to take this effect into account is, again, presented by Weidmann (1992). However, to calculate crowd density of a link, its geometry has to be taken into account, as discussed by Lämmel (2011).

21.4 Conclusions and Future Work

The multimodal contribution allows the tracking an agent's movement in detail, essential for studies related to topics like evacuations, e-bikes, car sharing or public transport. Experiments testing the implementation and demonstrating its capabilities are described by Dobler (2013).

An application's required level of detail strongly influences the modeling approach selection. A simple model including agents' age and gender, but not incorporating agent-agent interactions, might be detailed enough for some studies (e.g., e-bikes or public transport). However, for other studies, a more detailed model, also simulating agent interactions, might be necessary.

A first implementation of a pedestrian simulation module for MATSim, which also supports agent-agent interactions, was presented by Lämmel and Plaue (2014) introducing a force-base model. The agents' high-level planning (i.e., route and destination choice) was performed on a graph representing the transport system (e.g., a MATSim network), while the low level behavior (i.e., physical interaction between the participants) was simulated with a force-based model. Due to the intense computational effort of the underlying physical model, the scenario size was limited to a few thousand agents. An attempt to bypass this limitation was presented by Dobler and Lämmel (2012). They combined the force-based pedestrian simulation module with the multimodal link contribution, creating the opportunity to simulate large-scale scenarios, by staying highly resolved where needed and being more aggregated where possible.

CHAPTER 22

Car Sharing

Francesco Ciari and Milos Balac

22.1 Basic Information

Entry point to documentation:

<http://matsim.org/extensions> → carsharing

Invoking the module:

<http://matsim.org/javadoc> → carsharing → RunCarsharing class

Selected publications:

Ciari (2012)

22.2 Background

The basic carsharing idea is simple; a fleet of cars can be shared by several users, who can rent a car when needed, without having to own one. The possibility of renting short-term is the main difference from traditional car rentals. This basic concept can be implemented in various ways; in the last few years, several new business models have emerged on the market. From an operational perspective, there are three main variations:

- Round-trip based: Cars are parked at dedicated stations. They can be picked up from a station and left at the same station after use.
- One-way: Cars are parked at dedicated stations. They need to be picked up from a station and left at any station after use.
- Free-floating: Cars are parked in any parking slot within a defined service area. They can be picked up and left after use anywhere within this area.

How to cite this book chapter:

Ciari, F and Balac, M. 2016. Car Sharing. In: Horni, A, Nagel, K and Axhausen, K W. (eds.) *The Multi-Agent Transport Simulation MATSim*, Pp. 141–144. London: Ubiquity Press. DOI: <http://dx.doi.org/10.5334/baw.22>. License: CC-BY 4.0

From a transport planning perspective, the essential element of carsharing—the importance of its availability at precise points in time and space—does not fit with traditional models, which consider vehicle-per-hour flows. It is crucial to represent availability of vehicles at the local level, thus representing individual travel with high spatial and temporal resolution. At the same time, for the choice of using carsharing it is of fundamental importance how a trip/activity is embedded in the whole activity chain. This combination of features can be found in MATSim, which is therefore a suitable framework for carsharing modeling.

22.3 Modeling of Carsharing Demand in MATSim

Carsharing as a modal option in MATSim has been introduced in a simplified manner and only in its round trip-based version, as part of a dissertation work (Ciari, 2012). Several improvements have been introduced since then and all three main types of carsharing can now be simulated.

22.3.1 Round-Trip Based Carsharing

The use of round-trip carsharing by an agent in the simulation is modeled in the following steps:

1. The agent finishes his/her activity, finds the closest available car and reserves it (making it unavailable for other agents),
2. walks to the station where he/she has reserved a vehicle,
3. drives the car (interaction with other vehicles is modeled),
4. parks the car at the next activity.
5. After finishing his activity the agent takes the car and drives to the next activity.
6. Before reaching the last activity in the subtour, agent ends the rental and leaves the vehicle at the starting station, making it available to other agents,
7. walks to the activity, and
8. carries out the rest of the daily plan.

22.3.2 One-Way Carsharing

In the case of one-way carsharing, the steps are similar, but with few significant differences:

1. The agent finishes his activity, finds the closest station with an available car and reserves the vehicle (making it unavailable for other agents),
2. walks to the station where it has reserved the car (takes the car and frees a parking spot at the station),
3. finds the closest station to his destination, with a free parking spot and reserves it (making it unavailable for others),
4. drives the car to the reserved parking spot (interacting with other vehicles in the network),
5. parks the car on the reserved parking spot and ends the rental,
6. walks to the next activity, and
7. carries out the rest of the daily plan.

22.3.3 Free-Floating Carsharing

The use of free-floating carsharing by an agent is simulated using similar steps, but the rental ends with the end of one trip:

1. rent the nearest car,
2. walk from start activity to the rented car,
3. drive to the next activity (interaction with other vehicles are modeled),
4. park the car close to the next activity, and
5. end the rental (and make the car available for other rentals).

22.3.4 Generalized Cost of Carsharing Travel

The function representing generalized cost of travel for car sharing traveling from activity $q - 1$ to activity q is:

$$S_{trav,q,cs} = \alpha_{cs} + \beta_{c,cs} \cdot c_t \cdot t_r + \beta_{c,cs} \cdot c_d \cdot d + \beta_{t,walk} \cdot (t_a + t_e) + \beta_{t,cs} \cdot t \quad (22.1)$$

The same equation is used for all modeled forms of carsharing but the values of the parameters will be different. The first term α_{cs} is a constant which can be used as calibration parameter and will also be, generally, different for different types of carsharing (and for different context). The second and third terms refer to the time dependent and the distance dependent parts of the fee, respectively. t_r is the total reservation time and c_t represents the monetary cost for one hour reservation time. d is the total reservation distance and c_d is the marginal monetary cost for one kilometer travel. The parameter $\beta_{c,cs}$ represents the marginal utility of an additional unit of money spent on traveling with carsharing. The fourth term is the walk path to and from the station (access time t_a and egress time t_e) and is evaluated as a normal walk leg. The parameter $\beta_{t,cs}$ represents the marginal utility of an additional unit of time spent on traveling with carsharing, where t is the actual (in vehicle) travel time.

22.4 Carsharing Membership

Carsharing is a membership program. In order to access a specific carsharing service, individuals must become members of that carsharing program. A logit model has been estimated for Switzerland (Ciari and Weis, forthcoming) and implemented in MATSim as part of the carsharing module. The model variables are mainly individual socio-demographic characteristics. An important feature of the model, however, is that carsharing accessibility is explicitly considered, both from home and from work. Accessibility A of person p is calculated with the following formula:

$$A(n) = \ln \left(\sum_{s=1}^m X_s \cdot e^{-\beta \cdot d_{sh}} \right) + \ln \left(\sum_{s=1}^m X_s \cdot e^{-\beta \cdot d_{sw}} \right) \quad (22.2)$$

The weight parameter for distances is set to 0.2 as in Weis (2012), and more details on it are given below. Assuming m as the number of stations in the system, d_{sh} and d_{sw} , are calculated for each station as the distance between the station s and the home and work location of person n respectively; and X_s is the number of cars at station s . The model is not directly transferable to other regions but a different model can be easily implemented in the Java code created.

22.5 Validation

The simulation model has been calibrated to reproduce actual modal share for carsharing in the Zürich, Switzerland region. It was made using booking data from the Swiss operator Mobility. With the same data, the results were validated along several dimensions. Since Mobility offered only round-trip based carsharing until now, only this model could be validated. Dimensions included in the validation process were: distance from the last activity to the pick-up station, departure times, purpose of the rental (main purpose of the subtour) and temporal length of the rental.

22.6 Applications

After a long phase of creating and improving the module to simulate carsharing in all its forms, work has been recently carried out on concrete carsharing operations issues. Examples include evaluating the impact of introduction of a free-floating carsharing program in Berlin (Ciari et al., 2014) and Zürich (Ciari et al., forthcoming) on travel demand and investigating the relationship between demand and supply in both round-trip and one-way systems (Balac et al., 2015).

CHAPTER 23

Dynamic Transport Services

Michal Maciejewski

Entry point to documentation:

<http://matsim.org/extensions> → dvrp

Invoking the module:

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

Selected publications:

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

23.1 Introduction

The recent technological advancements in ICT (Information and Communications Technology) provide novel, on-line fleet management tools, opening up a broad range of possibilities for more intelligent transport services: flexible, demand-responsive, safe and energy/cost efficient. Significant enhancements can aid in both traditional transport operations, like regular public transport or taxis and introduction of novel solutions, such as demand-responsive transport or personal rapid transport. However, the growing complexity of modern transport systems, despite all benefits, increases the risk of poor performance, or even failure, due to lack of precise design, implementation and testing.

One solution is to use simulation tools offering a wide spectrum of possibilities for validating transport service models. Such tools have to model, in detail, not only the dynamically changing demand and supply of the relevant service, but also traffic flow and other existing transport services, including mutual interactions/relations between all these components. Although several approaches have been proposed (e.g., Regan et al., 1998; Barcelo et al., 2007; Liao et al., 2008;

How to cite this book chapter:

Maciejewski, M. 2016. Dynamic Transport Services. In: Horni, A, Nagel, K and Axhausen, K W. (eds.) *The Multi-Agent Transport Simulation MATSim*, Pp. 145–152. London: Ubiquity Press. DOI: <http://dx.doi.org/10.5334/baw.23>. License: CC-BY 4.0

Certicky et al., 2014), as far the author knows, no existing solutions provide large-scale microscopic simulation that include all the components above.

23.2 DVRP Contribution

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

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

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

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

23.3 DVRP Model

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

- one-to-many (many-to-one) and many-to-many topologies,
- multiple depots,
- dynamic requests,
- request and vehicle types,
- time windows for requests and vehicles,

- time-dependent stochastic travel times and costs, and
- network-based routing (including route planning, vehicle monitoring and diversion).

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

23.3.1 Schedule

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

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

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

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

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

23.3.2 Least-Cost Paths

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

¹ Package `org.matsim.contrib.dvrp.data`.

² Package `org.matsim.contrib.dvrp.schedule`.

³ Package `org.matsim.contrib.dvrp.tracker`.

to another in the network, or more precisely, from the end of one link end to the end of another link.

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

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

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

23.4 DynAgent

Contrary to the standard day-to-day learning in MATSim (but see also Section 97.3.5), in the DVRP contribution, each driver behaves dynamically and follows orders coming continuously from the dispatcher. The `DynAgent` class, along with the `org.matsim.contrib.dynagent` package, provides the foundation for simulating dynamically behaving agents. Although created for DVRP contribution needs, `DynAgent` is not limited to this context and can be used in a wide range of different simulation scenarios where agent dynamism is required.

`DynAgent`'s main concept assumes an agent can actively decide what to do at each simulation step instead of using a pre-computed (and occasionally re-computed; see 30.4.2) plan. It is up to the agent whether decisions are made spontaneously or (re-)planned in advance. In some applications, a `DynAgent` may represent a fully autonomous agent acting according to his/her desires, beliefs and intentions, whereas in other cases, it may be a non-autonomous agent following orders systematically issued from the outside (e.g., a driver receiving tasks from a centralized vehicle dispatching system).

23.4.1 Main Interfaces and Classes

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

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

At the higher level, the `DynAgent` dynamism results from the fact that dynamic activities and legs are usually created on the fly by the agent's `DynAgentLogic`; thus, the agent does not have to plan

⁴ Package `org.matsim.core.router`.

future actions in advance. When the agent has a roughly detailed legs and activities plan, he/she does not have to adhere to it and may modify his/her plan at any time (e.g., change the mode or destination of a future leg, or include or omit a future activity).

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

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

23.4.2 *Configuring and Running a Dynamic Simulation*

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

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

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

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

23.4.3 *RandomDynAgent Example*

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

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

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

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

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

23.5 Agents in DVRP

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

23.5.1 Drivers

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

23.5.2 Passengers

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

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

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

⁶ Package `org.matsim.contrib.dvrp.passenger`.

23.6 Optimizer

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

In the most basic case, an optimizer implements the `VrpOptimizer` interface⁷, that is, implements the following two methods:

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

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

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

23.7 Configuring and Running a DVRP Simulation

Like in within-day replanning (see Chapter 30), dynamic transport services are typically run with the DVRP contribution as a single-iteration simulation. Setting up and running such a simulation requires carrying out the following steps:

1. Create a Scenario (MATSim's domain data) and `VrpData` (VRP's domain data),
2. create a `VrpOptimizer`; this includes instantiation of a least-cost path/tree calculator, e.g., `VrpPathCalculator`, and

⁷ Package `org.matsim.contrib.dvrp.optimizer`.

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

3. call `DynAgentLauncherUtils.initQSim(Scenario scenario)` method to create and initialize a QSim; this includes creating a series of objects, such as an `EventManager`, `DynActivityEngine`, or `TeleportationEngine`.
4. When simulating passenger services, add a `PassengerEngine` to the QSim; this includes instantiation of a `PassengerRequestCreator` that converts calls/orders into `PassengerRequests`; otherwise (i.e., non-passenger services), add an appropriate source of requests to the QSim, either as a `MobsimEngine` or `MobsimListener`.
5. Then, add `AgentSources` to the QSim; for the `DynAgent`-based drivers, one may use a specialized `VrpAgentSource` and provide a `DynActionCreator`.⁹
6. run the QSim simulation, and
7. finalize processing events by the `EventManager`.

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

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

23.8 OneTaxi Example

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

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

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

23.9 Research with DVRP

Currently, the DVRP contribution is used in several research projects. Two of them focus on on-line dispatching of electric taxis in Berlin and Poznan (Maciejewski and Nagel, 2013b,c,a; Maciejewski, 2014). Another project deals with design of demand-responsive transport, where DVRP has been applied to the case of twin towns, Yarrawonga and Mulwala, described in Chapter 95 (Ronald et al., 2015, 2014). In a recently launched project, the DVRP contribution will be used for simulation of DRT services in three cities: Stockholm, Tel Aviv and Leuven.

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

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

SUBPART FIVE

Commercial Traffic

CHAPTER 24

Freight Traffic

Michael Zilske and Johan W. Joubert

24.1 Basic Information

Entry point to documentation:

<http://matsim.org/extensions> → freight

Invoking the module:

<http://matsim.org/javadoc> → freight → RunChessboard class

Selected publications:

Schröder et al. (2012); Zilske et al. (2012)

Various MATSim freight traffic modeling approaches have been implemented in recent years.

For Zürich, available origin-destination matrices for small delivery trucks and heavy trucks have been disaggregated Shah (2010). Data was taken from the KVMZH (Kantonales Verkehrsmodell Zürich) provided by Amt für Verkehr, Volkswirtschaftsdirektion Kanton Zürich (2011) and documented in Gottardi and Bürgler (1999). This special freight sub-population is restricted to route choice.

In South Africa, freight vehicles' plans were derived from GPS records of more than 30 000 commercial vehicles tracked over a 6-month period. Activity chains' extraction from raw GPS data was documented in Joubert and Axhausen (2011); the first joint private car and freight implementation appeared in Joubert et al. (2010). In Nagel et al. (2014), we used MATSim to evaluate the impact of a complex vehicle-type specific toll structure where sub-populations, including freight, have different time values.

The most sophisticated solution, however, was the introduction of carrier agents, described in the following section.

How to cite this book chapter:

Zilske, M and Joubert, J W. 2016. Freight Traffic. In: Horni, A, Nagel, K and Axhausen, K W. (eds.) *The Multi-Agent Transport Simulation MATSim*, Pp. 155–156. London: Ubiquity Press. DOI: <http://dx.doi.org/10.5334/baw.24>. License: CC-BY 4.0

24.2 Carriers

Until now, real-world scenarios set up with MATSim modeled freight traffic demand share by using plan sets with activities at the depot and pick-up and delivery locations, without variability in any dimension except route choice. We improved this situation by modeling freight vehicles as non-autonomous agents employed by, and serving the interests of, freight operators. Freight vehicle drivers' missing choice dimensions are then realized as logistics decisions made by the freight operators who employ them. In the freight transport sector, decisions are distributed among actors with different roles. Freight transport decisions include: lot-size choice, path-searches in logistical networks, vehicle choice and tour planning. A freight operator's planning problem is quite different from its passenger counterpart.

First, success of freight transport plans is not determined by the utility of time spent at activity locations, but rather by commercial success. Plans must fulfill customers' requirements, i.e., time windows and providing sufficient capacity at reasonable cost.

Second, freight operators often operate several vehicles and their options include rescheduling deliveries from one vehicle to another or even changing the number of vehicles used.

Thus, a new software layer populated by *carrier agents* was introduced into the simulation. Each carrier agent represents a firm with a vehicle fleet, depots and contracts. Contracts determine type and quantity of goods to be carried and contains the respective origin and destination as well as pick-up and delivery time windows.

The carrier agent's plan contains a tour schedule for each fleet vehicle, containing planned pick-up, delivery or arrival times at customer locations and a route through the physical network. In our basic model, all vehicle schedules of a carrier begin and end at one of its depots. When a simulation scenario is initialized, the carrier agents build a schedule for each of their vehicles, including a route through the transport network, with pick-up and delivery activities corresponding to their contracts. At the interface between the freight operator plans and the mobility simulation, the set of routed vehicles from each carrier plan is injected into the traffic demand as individual driver agents, where they move through the traffic system along with passenger vehicles. While executing their plans, the freight driver agents report their shipment-related activities back to the carrier.

When all plans have been executed, agents evaluate the success of their plan. The carrier agents use a custom utility function capturing their economic success. Their cost is calculated as a sum of vehicle-dependent distance and time costs incurred by scheduled vehicles, as well as some individual fixed costs, plus penalties incurred by missed time windows.

Finally, carrier agents create new plans to improve their performance in the next iteration. For instance, a time-dependent vehicle routing heuristic can be plugged in to replan vehicle schedules. Shipments can be switched between vehicles, or an entire vehicle added or removed. During repeated executions of their plans, passengers as well as carriers gain experience from the transport system. The carriers experience congestion and other disturbances in the traffic system when they incur a higher cost through longer vehicle usage, or by penalizing missed pick-up and delivery times.

The planning algorithms themselves are implemented in the project *jsprit*, a library separate from MATSim. In the replanning phase of each iteration, *jsprit* is called and replans the carrier plans.

The model is described in a paper by Schröder et al. (2012). For more details about the implementation, as well as more references, see the technical report by Zilske et al. (2012).

CHAPTER 25

WagonSim

Michael Balmer

25.1 Basic Information

Entry point to documentation:

<http://matsim.org/extensions> → wagonSim

Invoking the module:

<http://matsim.org/javadoc> → wagonSim → RunWagonSim class

Selected publications:

-

25.2 Summary

The wagonSim contribution allows use of MATSim's route-optimization process to find optimal paths for rail-based freight wagons in a given rail-based freight infrastructure.

The network links, here, define the rails, nodes define train stations and schedule transit stops define train station stopping points. Freight locomotives are driven by a strictly fixed schedule, where each locomotive is given as a single transit line with a single transit route and a single departure. Freight wagons correspond to agents with a given origin and destination (single trip agents). Routing takes various constraints into account, i.e., a minimum shunting time while switching locomotives and maximum freight train weight and length; it also differentiates between locomotive stops for shunting and stops only for waiting (without shunting possibility).

WagonSim contribution is based on specialized input data. The first step converts input data into MATSim formats (scenario data). In a second step, it allows one to manually adapt the scenario for different parametrization of train stops, shunting stations, minimum shunting times and

How to cite this book chapter:

Balmer, M. 2016. WagonSim. In: Horni, A, Nagel, K and Axhausen, K W. (eds.) *The Multi-Agent Transport Simulation MATSim*, Pp. 157–160. London: Ubiquity Press. DOI: <http://dx.doi.org/10.5334/baw.25>. License: CC-BY 4.0

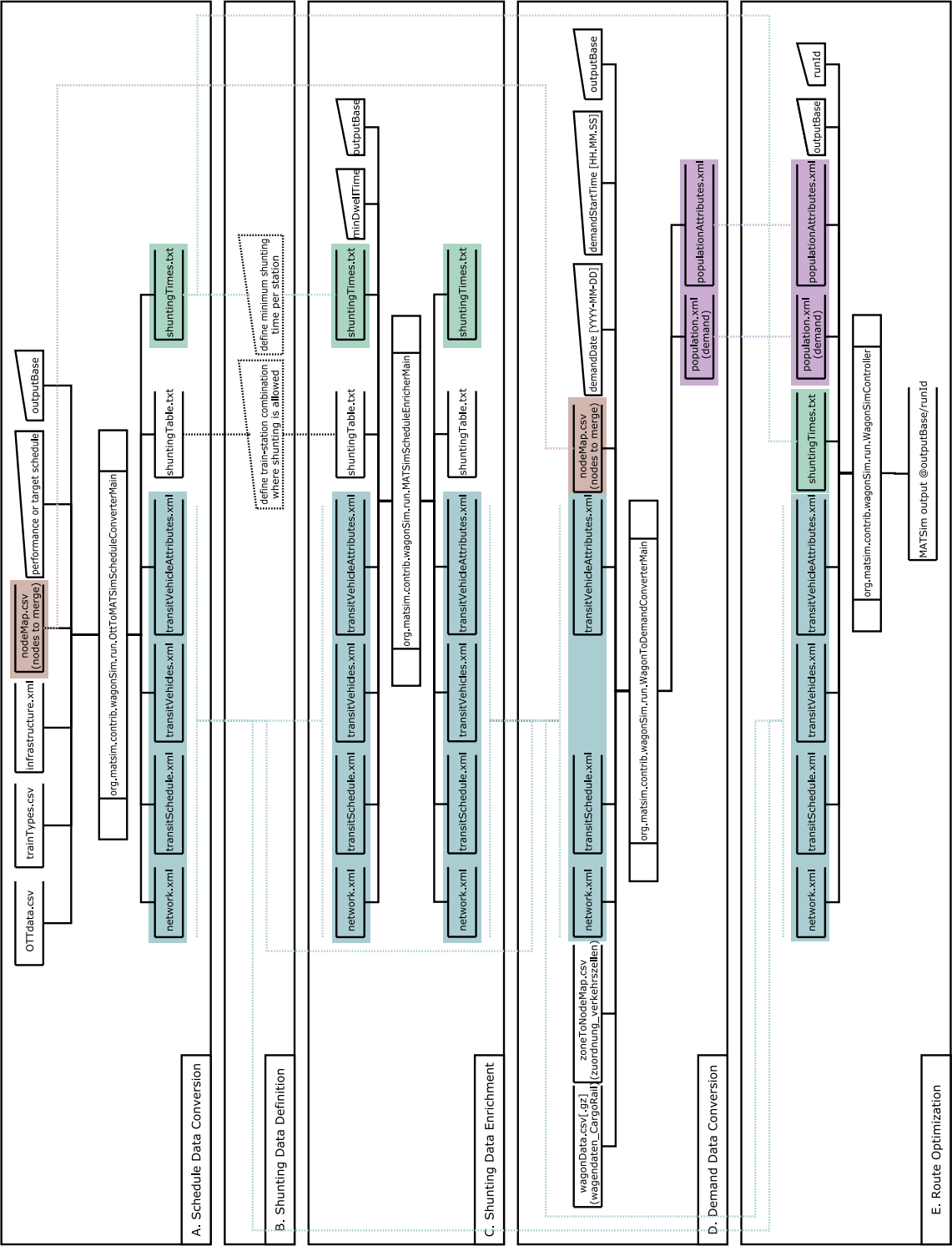


Figure 25.1: WagonSim process chain.

dwel times of trains at stops. The third step sets up route optimization configuration and runs the MATSim optimization cycle.

As shown at <http://www.matsim.org/docs/extensions/wagonSim> and in Figure 25.1, data conversion and WagonSim execution is composed of five stages, described in more detail at above referenced url:

- A)* schedule data conversion,
- B)* shunting data definition,
- C)* shunting data enrichment,
- D)* demand data conversion, and
- E)* route optimization.

WagonSim contribution has been applied to ETH (Eidgenössische Technische Hochschule), IVT (Institut für Verkehrsplanung und Transportsysteme – Institute for Transport Planning and Systems) Transport Systems group projects.

CHAPTER 26

freightChainsFromTravelDiaries

Kai Nagel

Entry point to documentation:

<http://matsim.org/extensions> → freightChainsFromTravelDiaries

Invoking the module:

Currently not possible.

Selected publications:

Schneider (2011)

Sebastian Schneider has done a Ph.D. dissertation about generating freight vehicle chains by essentially re-sampling the information contained in the German survey KiD (Kraftfahrzeugverkehr in Deutschland) (Steinmeyer and Wagner, 2005). Since the KiD is essentially an activity-based travel diary, the method should also be applicable to other situations. Since Sebastian has left science for the time being, he allowed us to take his code and integrate it into the repository, under the GPL (GNU General Public License). For the time being, it will just “sit” here until someone attempts to make it work.

How to cite this book chapter:

Nagel, K. 2016. freightChainsFromTravelDiaries. In: Horni, A, Nagel, K and Axhausen, K W. (eds.) *The Multi-Agent Transport Simulation MATSim*, Pp. 161–162. London: Ubiquity Press. DOI: <http://dx.doi.org/10.5334/baw.26>. License: CC-BY 4.0

SUBPART SIX

Additional Choice Dimensions

CHAPTER 27

Destination Innovation

Andreas Horni, Kai Nagel and Kay W. Axhausen

27.1 Basic Information

Entry point to documentation:

<http://matsim.org/extensions> → locationchoice

Invoking the module:

<http://matsim.org/javadoc> → locationchoice → RunLocationChoiceBestResponse, RunLocationChoiceFrozenEpsilons classes

Selected publications:

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

27.2 Introduction

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

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

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

As well as considering competition for road infrastructure, the destination choice module can also be configured for activities infrastructure (for example, at shopping malls' parking lots) as shown in Section 27.3.5 and by Horni et al. (2009).

How to cite this book chapter:

Horni, A, Nagel, K and Axhausen, K W. 2016. Destination Innovation. In: Horni, A, Nagel, K and Axhausen, K W. (eds.) *The Multi-Agent Transport Simulation MATSim*, Pp. 165–174. London: Ubiquity Press. DOI: <http://dx.doi.org/10.5334/baw.27>. License: CC-BY 4.0

27.3 Key Issues in Developing the Module

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

The main component however, is a mechanism to generate consistent random draws over iterations necessary to include the objective function's error terms (see next Section 27.3.1). This mechanism is also applicable to other choice dimensions.

27.3.1 Error Terms

As described in Chapter 49, MATSim—as a utility-maximizing model—is related to the discrete choice framework, meaning that this framework can productively guide the MATSim utility function specification. Utility in discrete choice models is composed of a deterministic part and a random error term representing the unobserved heterogeneity, i.e., it subsumes, both truly, i.e., inherently random, decisions and the modeler's missing knowledge about the choice and its context.

In MATSim, the utility function for route, mode and time innovation does not contain an explicit random error term (yet). This is at least partially compensated through replanning stochasticity, in Chapter 49 denoted by the scale parameter μ and η . An example for this might be: route and time choices are usually subject to significant competition. The co-evolutionary algorithm of MATSim, detailed below, essentially assigns the resources in a random manner to the persons. For example, two identical persons may end up with different routes, according to the order in which they undergo the replanning. Essentially, this means that an (implicit) random term is present in the choice making.

The above, however, does not add enough unobserved heterogeneity to destination choice. Further problems might, or might not, appear when trying to interpret this randomness, since it is added implicitly and somewhat unsystematically. Thus, an explicit random error term ε_{nlq} for every person n , alternative ℓ and activity q , held stable over the iterations, is added to the scoring function during the running of the destination innovation module (Horni, 2013). Research about the necessity of error terms for the remaining choice dimensions is required, as discussed in Section 97.4.6.

27.3.2 Quenched Randomness

Due to random error terms, discrete choices are quantified by probabilities; for example, for the logit model, as $p_{nlq} = \exp(V_{nlq}) / \sum_{j \in L} \exp(V_{njq})$, where V_{nlq} is person n 's systematic utility of alternative ℓ for activity q . When drawing from the distribution specified by p_{nlq} for a population, the aggregate choices are reproduced. This is basically also true when applied in iterative frameworks. However, iterative frameworks are usually associated with some kind of learning or relaxation mechanism, which is heavily distorted by repeatedly and randomly drawing from p_{nlq} in every iteration. In this case, the ε_{nlq} effectively fluctuate from iteration to iteration, which is disastrous for the algorithm's convergence and behaviorally implausible.

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

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

This includes, via the systematic part V_{nlq} , the disutility of traveling to destination ℓ for activity q .

As stated above, random error terms must remain the same over the iterations (also discussed in Chapter 49). In physics, this approach would be called “quenched” (sometimes also “frozen”) randomness; all randomness is computed initially and then attached to particles or destinations, rather than instantaneously generating it, which would be called “annealed” randomness. Two natural approaches for implementing quenched randomness are as follows:

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

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

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

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

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

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

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

27.3.3 Search Space Construction and Evaluation

MATSim destination innovation is based on best-response, rather than random mutation; in every iteration, the best current alternative, *including* the $\varepsilon_{n\ell q}$, is chosen. This works as long as inter-iteration changes are small, which usually happens, given by the relatively small share of agents who re-plan. The best-response approach is adopted due to the usually huge number of alternatives in combination with the search space characteristics. The discrete search landscape is characterized by random noise, because error terms are not spatially correlated (see Figure 27.1(a)). For such problems—as opposed to continuous landscapes (see Figure 27.1(b))—efficient search methods, such as local search methods, generally do not work.

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

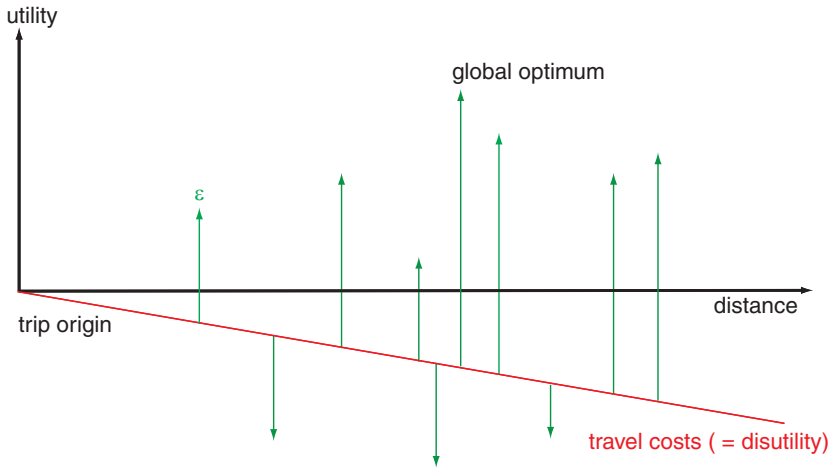
This search space reduction saves a great deal of computation time; however, it is still unfeasible and further speed-ups are necessary. Most computation time is due to travel time calculation, i.e., due to routing, for evaluation of the alternatives in the search space. To reduce these huge routing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

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

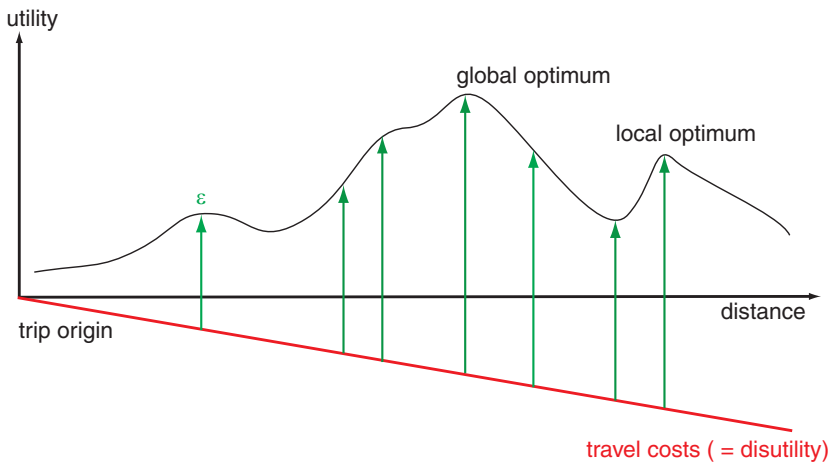
27.3.4 Destination Choice Set Specification

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

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



(a) Uncorrelated error terms.



(b) Spatially correlated error terms.

Figure 27.1: Search space: The search algorithm must be able to handle correlated, as well as uncorrelated, error terms as given by the MNL model. Local search methods, such as hill-climbing algorithms are only able to handle continuous search spaces; thus, for situation (a), a best-response global search algorithm is required.

(1979); Burnett (1980); these integrate the choice set formation step into the estimation procedure by jointly estimating choice set selection and selection of a particular alternative of this choice set (Manski, 1977; Ben-Akiva and Boccara, 1995). Probabilistic choice set formation is conceptually appealing; choice sets are, in principle, not restrained a priori by exogenous criteria, as in standard choice set specification. However, the procedure is generally associated with combinatorial complexity, making it computationally intractable. As a consequence, practical approaches also require mechanisms to reduce complexity of the choice set specification problem (e.g., Ben-Akiva and Boccara, 1995, p.11). Zheng and Guo (2008), for example, make the moderate assumption of continuous store choice sets (i.e., sets without “holes”) around the trip origin, while Ben-Akiva and Boccara (1995)’s random-constraints model exploits additional information on alternatives’ availability for individuals.

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

27.3.5 Facility Load

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

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

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

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

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

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

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

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

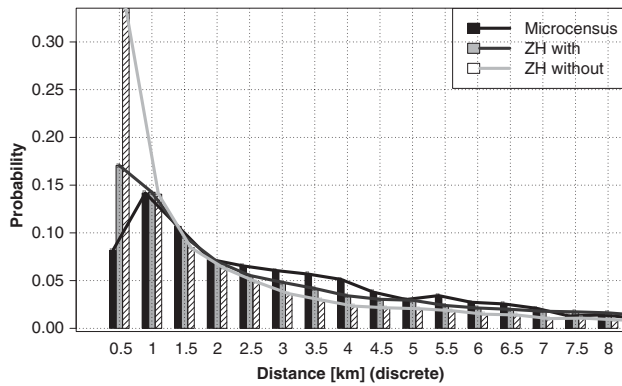
As expected, the constrained model improves result quality by reducing the number of implausibly overcrowded activity locations.

27.4 Application of the Module

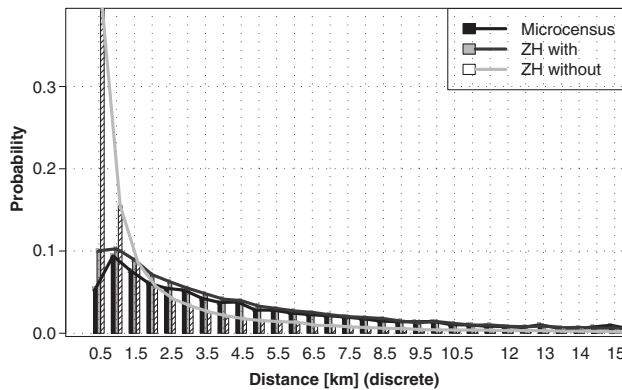
The destination innovation module has been successfully applied for the Zürich scenario (Chapter 56), as reported in Horni (2013, p.99), for the Tel Aviv model (see Chapter 91) and for the MATSim 2030 project. Figure 27.2 and Figure 27.3 show that, through error term scaling, distance distributions can be nicely fitted, decreasing count data error.

27.5 The Module in the MATSim Context

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



(a) Shopping trips.



(b) Leisure trips.

Figure 27.2: Error term runs for the Zürich scenario.

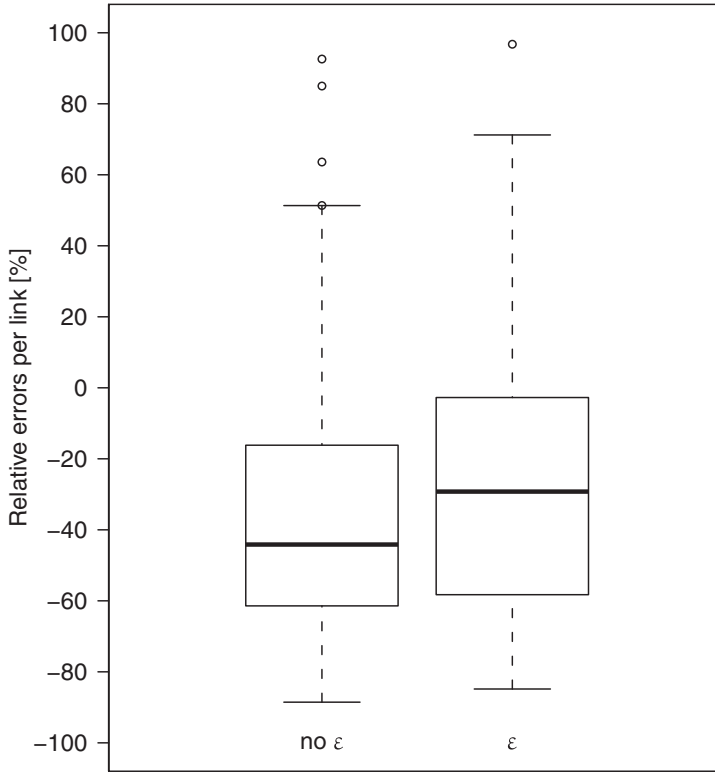


Figure 27.3: Daily traffic volumes for 123 links compared to traffic counts. Per link k the relative error is used, i.e., $(vol_{simulated,k} - vol_{counted,k})/vol_{counted,k}$.

it out. Whether the standard utility function might also profit from the innovations made for this module should be a topic for future research.

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

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

27.6 Lessons Learned

Two interesting lessons were learned while developing the destination innovation module: first, a lesson on preferences and space interdependence and the necessity to evaluate them in combination. When looking at distance distributions (e.g., Figure 27.2) one might think that the functional form directly represents the preferences, but this is not necessarily the case. In our simulations, it is the result of a *linear* travel disutility, but applied in geographic space, where number of opportunities increases with the *square* of the radius, in other words, with the *square* of travel distance. A similar emergent effect appears when scaling random error terms. Although both negative and

positive error terms are enlarged and the average remains stable, distribution gets more skewed toward the tail; for agents' choices, maximum values—not average values—are relevant.

The second lesson concerns simulation results' variability. Although random elements are not present only in destination choice, it was the largest contributor of endogenous variability when it was developed, necessitating the experiments presented by Horni et al. (2011a) (see also Section 48.4).

27.7 Further Reading

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

CHAPTER 28

Joint Decisions

Thibaut Dubernet

28.1 Basic Information

Entry point to documentation:

<http://matsim.org/extensions> → socnetsim

Invoking the module:

<http://matsim.org/javadoc> → socnetsim → `RunExampleSocialSimulation` class

Selected publications:

Dubernet and Axhausen (2013), Dubernet and Axhausen (2014)

This chapter describes the extension of MATSim to consider what we call *joint decisions*. Section 28.2 explains what we call a joint decision, and gives an overview of why such processes are important in transportation. Section 28.3 then presents concepts to model this behavior, a generalization of the MATSim algorithm to search for solutions to the *joint planning problem*, and gives technical insights on how this implementation could be achieved, given the MATSim software architecture.

28.2 Joint Decisions and Transport Systems

28.2.1 Motivation

In recent years, there has been a growing interest in the social dimension of travel and how travel decisions are influenced, not only by the global state of the transportation system, but also by joint decisions and interactions with social contacts.

How to cite this book chapter:

Dubernet, T. 2016. Joint Decisions. In: Horni, A, Nagel, K and Axhausen, K W. (eds.) *The Multi-Agent Transport Simulation MATSim*, Pp. 175–182. London: Ubiquity Press. DOI: <http://dx.doi.org/10.5334/baw.28>. License: CC-BY 4.0

A very active field of research is the study and modeling of intra-household interactions and joint decision-making, often using the classical random utility framework extended to group decision-making. Examples of household scheduling models include: Zhang et al. (2005, 2007); Kato and Matsumoto (2009); Bradley and Vovsha (2005); Gliebe and Koppelman (2005, 2002); Ho and Mulley (2013); Vovsha and Gupta (2013). Most of those models are specific to given household structures; in particular, separate models need to be estimated for different household sizes.

Another class of approaches, more oriented toward multi-agent simulation than analysis, is the use of optimization algorithms to generate households plans. These algorithms handle the household scheduling problem by transforming it into a deterministic utility maximization problem. Contrary to the previously presented approaches, those alternatives do not lead to the estimation of a model against data. Examples of approaches rooted in operations research include: Recker (1995), for which Chow and Recker (2012) designed a calibration method, or Gan and Recker (2008). Another attempt to generate plans for households uses a genetic algorithm, building on a previous genetic algorithm for individual plan generation (Charypar and Nagel, 2005; Meister et al., 2005), using a joint utility. Finally, Liao et al. (2013) formulate the problem of creating schedules for two persons traveling together as finding the shortest path in a “supernetwork”, but note that their model is specific to the two person problem and that extension to larger numbers of agents may prove to be computationally expensive. All those approaches remained experimental, and were not integrated into multi-agent simulation tools.

Another class of methods aiming at multi-agent simulations is constituted rule based systems, which use heuristic rules to construct household plans, such as Miller et al. (2005); Arentze and Timmermans (2009).

Other authors have investigated the role of more general social networks on travel. One of the main incentives to conduct such studies comes from the continuous increase of the share of leisure purpose trips (Schlich et al., 2004; Axhausen, 2005). This trend represents a challenge for travel behavior modeling, as those trips are much more difficult to forecast than commuting trips; they are performed more sporadically and data from such trips is much more difficult to collect—particularly concerning location and event attributes, necessary to make models that are more than just random noise. A better understanding of how leisure trip destination choices are made is essential to improve the accuracy of those forecasts.

Various studies have been conducted to confirm that making social contacts is an important factor in leisure trip destination choice, or activity duration choice. Examples of empirical work include: Carrasco and Habib (2009); Habib and Carrasco (2011) or Moore et al. (2013). All these studies show strong influence of social contacts on the spatial and temporal distribution of activities. In a simulation experiment, Frei (2012) demonstrated that considering social interactions in leisure location choice helps increase the accuracy of predicted leisure trip distance distribution.

Another field of empirical research studies the spatial characteristics of social networks. For instance, Carrasco et al. (2008) studied the relationship between individual's socioeconomic characteristics and the spatial distribution of their social contacts. This kind of empirical work allows specification and estimation of models able to generate synthetic social networks, given sociodemographic attributes and home location. An example of such a model, based on the results of a survey in Switzerland, can be found in Arentze et al. (2012). This kind of model is essential if one wants to include social network interactions in microsimulation model.

This integration of social networks in multi-agent simulation frameworks has already been attempted by other authors. Due to their disaggregated description of the world, such models are particularly well-suited to complex social topologies representation. Han et al. (2011) present experiments using social networks to guide activity location choice set formation in the FEATHERS (Forecasting Evolutionary Activity-Travel of Households and their Environmental Repercussions) multi-agent simulation framework. Using a simple scenario with 6 agents forming a *clique*, they consider the influence of various processes like information exchange and adaptation to the behavior of social contacts to increase the probability of an encounter. They do not, however, represent

joint decisions, such as the scheduling of a joint activity. The same kind of processes have been investigated by Hackney (2009), using more complex network topologies (within the MATSim framework) used in this paper. Ronald et al. (2012); Ma et al. (2011, 2012) present agent based systems, which integrate joint decision-making mechanisms, based on rule based simulations of a bargaining processes. They are not yet integrated into any operational mobility simulation platform.

Those remarks point the need to include explicit coordination in multi-agent simulation platforms.

28.2.2 The Joint Planning Problem

Here, we present a simulation framework able to represent *joint decisions*: that is, behavior requiring *explicit* coordination between individuals—such as shared rides, social activities or intra-household task allocation. The basic idea is that social contacts will make such a joint decision if it results in an improvement in the satisfaction of all participants. Modeling the interaction of individuals with possibly conflicting objectives has been the subject of game theory for decades, making this theoretical framework particularly well suited for the problem at hand.

Interestingly, game theoretic view of transportation systems has been popular since the seminal work of Wardrop (1952). The essential underlying concept is a view of the transportation system as a set of shared resources (road space, public transport vehicle seats...), for which individuals compete; individuals in the population try to maximize their own satisfaction, given the resources left available by others. Game theory studies *solution concepts* for such strategic interactions. A game theoretic solution concept is a definition of which states are *equilibria*: that is, *stable* under assumption of rationality—a state is considered stable if no agent/player has an incentive to change its behavior. The static, trip-based approach of Wardrop (1952) has been refined and extended with time. In particular, the equilibrium idea can be quite readily transferred to the *activity based* framework: individuals try not only to optimize their trips, but their whole day. This is, in particular, the approach of MATSim (Axhausen, 2006; Nagel and Flötteröd, 2012).

Most solution concepts in transportation are akin to the Nash equilibrium: a state where no individual can improve its satisfaction by *unilaterally* changing its behavior. This kind of solution concept does not allow to represent joint decisions. This can be illustrated by a classical game, called the *House Allocation Problem* (Schummer and Vohra, 2007). This game consists of n players and n houses. Moreover, each player has its individual ordering of the houses, from the most preferred to the least preferred, and players prefer being allocated alone to any house rather than to a house occupied by someone else. The strategy of a player centers around the house where the player chooses to live.

An interesting feature of this game is that any one-to-one allocation of players to houses is a Nash Equilibrium; no player can improve its payoff by *unilaterally* changing its strategy, as it would require choosing an occupied house. This result, however, contradicts basic intuition about the stability of such an allocation. In this particular case, a more realistic solution concept is the *Absence of Blocking Coalition*; given a one-to-one allocation of houses to players, a blocking coalition is a set of players which could all be better off by reallocating their houses among themselves. It should be noted that both solution concepts correspond to rational agents, i.e., agents having a preference ordering over outcomes. The only difference lies in the degree of communication allowed.

In the activity-based framework, this solution concept naturally becomes what we define as the *Absence of Improving Coalition* solution concept. An improving coalition for a given allocation of daily plans is a set of social contacts who can all feel themselves to be better off by *simultaneously* changing their daily plan—for instance, by switching from separate dinners at home to a joint dinner at a restaurant. The simulation of joint decision consists of searching an allocation of daily plans without such coalitions.

28.3 A Solution Algorithm for the Joint Planning Problem: A Generalization of the MATSim Process

28.3.1 Algorithm

Given this theoretical framework, one needs to design and implement an algorithm to search for allocations of daily plans to individuals that satisfy this solution concept. This implementation consists of two groups of components:

1. A Controller that implements the extension of the MATSim co-evolutionary algorithm, outlined hereafter. It is implemented in a modular fashion, to be easily adapted to the specific need of different simulation scenarios and
2. specific implementations of the modular components, namely replanning strategies and scoring functions, to allow explore the set of possible *joint plans* and representations of possible preferences specific to joint decisions.

Controller The MATSim framework provides a Controller to build and configure co-evolutionary algorithms, where agents each optimize their plan given the (evolving) state of the transport system.

Unfortunately, this approach makes choices of agents independent—which, of course, goes against the simulation of *joint decisions*. To implement an algorithm searching for states without blocking coalitions, one needs a way to represent the influence of explicit coordination on daily plan utility. This is solved by including *joint plans* constraints. A joint plan is a set of individual plans executed simultaneously. Different copies of the same individual plan can be part of different joint plans—for instance, an agent might go to a given restaurant alone, with members of its household or with a group of friends. The score of the different copies will take into account the influence of the joint plan to which it pertains. Those joint plan constraints are included using heuristic rules, applied after mutation operators are applied, and are classified as strong or weak constraints—weak constraints are considered when selecting plans for execution, but are allowed to be broken when merely selecting plans for mutation. They are then part of the evolution process. In the current application, the heuristic rules consist of joining newly created plans with joint trips (strong), or with leisure activities at the same location at the same time (weak).

To allow handling joint plans, replanning needs to be performed for groups of agents. This is straightforward for households; all agents of the same household are always handled as a single group. For more general social networks, agents are handled with all agents with whom they have a joint plan, plus some social contacts with whom new joint plans can be created.

For each group, two actions are then possible. For most groups, an allocation of existing plans—fulfilling the joint plans constraints—is selected for execution. Based on plan scores, randomized by adding an extreme value distributed error term, an algorithm inspired by the “Top Trading Cycle” algorithm used for the “House Allocation Problem” (Schummer and Vohra, 2007) searches for an allocation without improving coalitions.

For the other groups, a plan allocation is selected and copied. The copied plans then undertake mutation, to make the agents explore new alternative joint plans. Which mutation is performed determines which alternative plans will be tried out by the agent.

Agents have a limited memory size, keeping by default at most three plans per joint plan composition, and ten plans in total. If this limit is exceeded, one should keep the plans which have the highest probability of creating improving coalition: that is, preferable to the other plans in the agent’s memory. To this end, a lexicographic ordering is used; the process removes the joint plan maximizing the number of individual plans which are the worst of the agents’ memories. If several joint plans have the same number of worst plans, the process chooses among them to find the joint plan which maximizes the number of second worst plans, and so on, until the “worst” joint plan is unique. When the overall maximum number of plans in the memory of an agent is reached, the worst individual plan for this agent is removed along with plans of other agents of the same joint

plan. Each agent keeps at least one plan that is not part of a joint plan, as there might otherwise be no state without blocking coalitions. Agents are parsed in random order, to avoid the emergence of “dictators” over iterations, whose worst plan would always be removed, even if it is the only “bad” plan of a joint plan.

Though those selection operators seem to be in accordance with the chosen solution concept, it is difficult, if not impossible, to prove that the process will actually converge towards the state searched. As noted by Ficici et al. (2005), when they perform a theoretical analysis of different selection methods in a co-evolutionary context, “co-evolutionary dynamics are *notoriously complex*. To focus on our attention on selection dynamics, we will use a simple evolutionary game-theoretic framework to eliminate *confounding factors* such as those related to genetic variation, noisy valuation, and finite population size”. Those “confounding factors” can, however, not be eliminated from an actual implementation of a co-evolutionary algorithm; rigorously proving that a given algorithm actually implements a specific solution concept is very tedious, if not impossible.

With iterations, agents build a choice set of daily plans that becomes better and better, given the actions of the other agents. However, the presence of a large group of agents with plans resulting from random mutation creates noise, not only for the analyst looking at the output of the simulation, but for the agents themselves when they compute their score plans. To solve this issue, when the system reaches a stable state, agents stop performing mutation, and select plans only from their memory for a given number of iterations, using the absence of improving coalition with randomized scores. This ensures that the selected plans are the result of a behavioral model, rather than the result of random mutation operators.

28.3.2 Technical Considerations on the Implementation

As highlighted in Chapter 45, the preferred way to add new behaviors to the MATSim software is by designing *pluggable* elements, that can be added to a Controller from a configuring “script”.

This modular approach works well in most of the cases used and makes it possible to combine different elements and design highly specific runs. There is, however, an element that one cannot modify this way: the general form of the evolutionary process. This process is exactly what has to be modified to include joint decisions—this section focuses on the challenges and solutions to undertaking such a major modification, as a reference from developers facing this exact problem.

The one important modification of the process: in the standard MATSim process, *replanning* is performed independently for each agent, whereas for joint decisions, agents must be replanned *as groups*: selection of plans needs to fulfill *joint plan constraints* and is performed using the group-level “absence of improving coalition” criterion, and mutation operators are allowed to work on several plans at the same time, for instance to insert *joint trips*, or select the location of a *joint activity*.

Doing so requires the replacement of the `ReplanningListener`, that is, the element responsible for managing the whole replanning step. This can only be done by implementing a separate Controller.

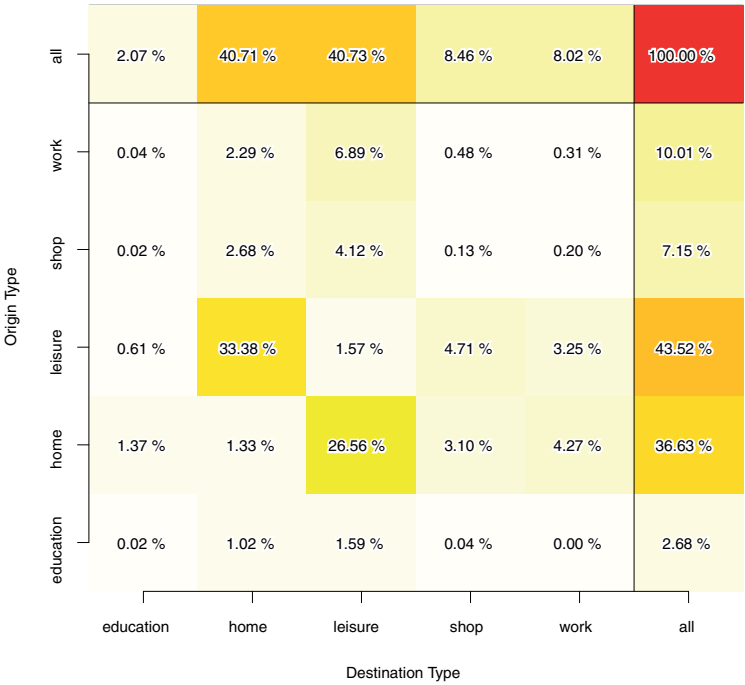
Modularity was kept as high as possible, in particular by providing standard ways to use the default individual-based replanning modules from this new element.

28.4 Selected Results

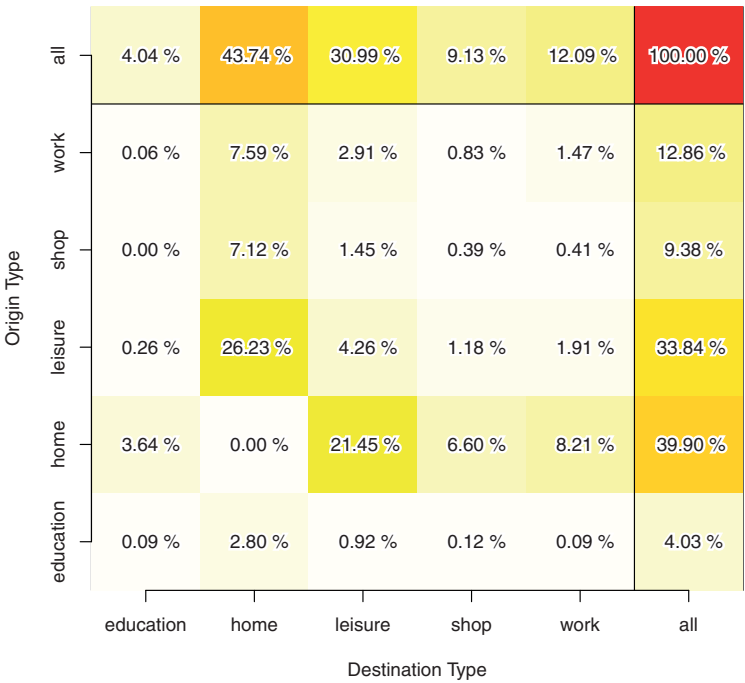
This section presents a few simulation results demonstrating how the approach can help improve simulation results. It uses a scenario using 2010 data, with a leisure contacts network generated using the approach of Arentze et al. (2013).

Specific replanning modules include: inclusion and removal of joint trips (by joining existing trips), and joint location choice for leisure activities. A specific scoring term is added to consider the preference for *joint activities*; individuals want to perform leisure activities with at least one social contact. Leisure time passed without any contact is penalized.

Figure 28.1 presents the repartition of “car passenger” trips by purpose, in the simulation as well as the Swiss National Travel Survey. The simulation is able to reflect the fact that most trips are



(a) Simulation.



(b) National Travel Survey.

Figure 28.1: Share of passenger trips by purpose.

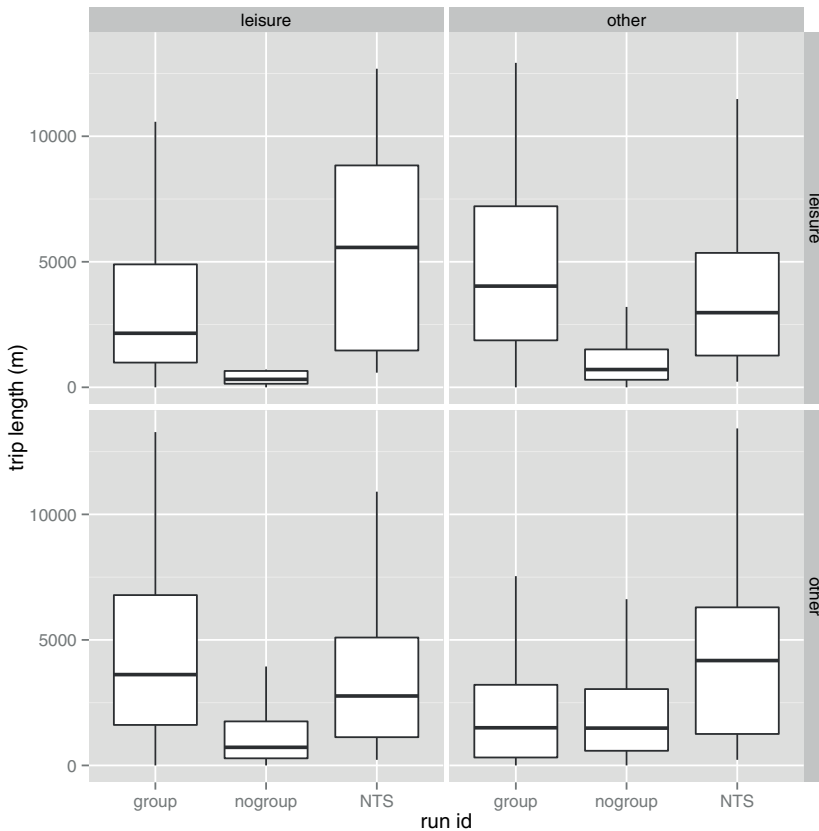


Figure 28.2: Car passenger travel distance to leisure activities.

performed for *leisure purposes*. Figure 28.2 shows the distance distribution of *car passenger* trips by purpose, with preference for group activities enabled or not, as well as in the “Swiss National Travel Survey”. The preference for joint activities certainly encourages individuals to travel together for leisure, without waiting for each other, resulting in distance distributions much closer to the “Swiss National Travel Survey” data with this parameter than without.

28.5 Further Reading

The work presented in this chapter has been described in more detail in various other papers. Dubernet and Axhausen (2013) presents an early stage of the algorithm, applied to a toy scenario. Dubernet and Axhausen (2014) provides more theoretical ground, making more explicit reference to game theory and compares two *solution concepts* for solving joint planning problems in the household case: first, the absence of improving coalition presented here and second, a “joint utility” formulation, well-represented in literature. Dubernet and Axhausen (forthcoming) presents a validation of the model for the household case, using a Zürich scenario. An independent approach to model household choices developed for the Baoding scenario is presented in Chapter 61.

CHAPTER 29

Socnetgen

Kai Nagel

29.1 Basic Information

Entry point to documentation:

<http://matsim.org/extensions> → socnetgen

Invoking the module:

<http://matsim.org/javadoc> → socnetgen → RunErgmSimulator class

Selected publications:

Illenberger (2012)

29.2 Summary

This package contains algorithms to generate social networks that may be used on top of the MATSim population. It pre-dates the work by Dubernet presented in Chapter 28. The approach in socnetgen is much more lightweight than that of Chapter 28, but it also does nothing beyond just generating the social network according to given statistical criteria.

How to cite this book chapter:

Nagel, K. 2016. Socnetgen. In: Horni, A, Nagel, K and Axhausen, K W. (eds.) *The Multi-Agent Transport Simulation MATSim*, Pp. 183–184. London: Ubiquity Press. DOI: <http://dx.doi.org/10.5334/baw.29>. License: CC-BY 4.0

SUBPART SEVEN

Within-Day Replanning

CHAPTER 30

Within-Day Replanning

Christoph Dobler and Kai Nagel

30.1 Basic Information

30.1.1 Implementation Alternative 1

Entry point to documentation:

<http://matsim.org/extensions> → withinday

Invoking the module:

<http://matsim.org/javadoc> → tutorial → RunWithinDayExample class

Selected publications:

See Section 30.4.2.

30.1.2 Implementation Alternative 2

Entry point to documentation:

<http://matsim.org/extensions> → withinday

Invoking the module:

<http://matsim.org/javadoc> → tutorial → RunOwnMobsimAgentUsingRouter class

Selected publications:

See Section 30.4.3.

How to cite this book chapter:

Dobler, C and Nagel, K. 2016. Within-Day Replanning. In: Horni, A, Nagel, K and Axhausen, K W. (eds.) *The Multi-Agent Transport Simulation MATSim*, Pp. 187–200. London: Ubiquity Press. DOI: <http://dx.doi.org/10.5334/baw.30>. License: CC-BY 4.0

30.2 Introduction

In recent years, transport planning and traffic management interest in unforeseeable, or only partially foreseeable events within scenarios has increased. Partially foreseeable events often occur with taxis and car sharing. For example, agents with a planned taxi trip cannot know in advance which taxi will be available when they need one. When using car sharing, an agent might walk to the car sharing station and check whether a car is available or not. If it is not, the agent could either decide to wait, or change its plan and switch to another transportation mode. Road accidents, terrorist attacks or disasters such as earthquakes are examples of completely unpredictable events.

As discussed earlier, traditional simulation approaches (used in default-MATSim) calculate demand-supply equilibria using an iterative process. There, it is assumed that a typical situation is simulated where agents can rely on their experience from comparable situations, like previous iterations. Applying an iterative approach to a scenario with unexpected events results in problems like illogical agent behavior, producing false results. In the next section, these problems, as well as an alternative simulation approach, are presented. On one hand, this approach—called within-day replanning—simulates only a single iteration, avoiding problems resulting from an iterative simulation process. On the other hand, this approach does require a more detailed behavioral model for the agents. Subsequently, using MATSim as a base, the iterative approach is discussed, followed by two different implementations of the within-day replanning approach into the framework, including discussions of the technical implementations.

30.3 Simulation Approaches

30.3.1 Iterative Simulation Approaches

An iterative day-to-day replanning approach is appropriate as long as the scenario describes a *typical* situation or day. For such scenarios, it is feasible to assume that agents are familiar with typically occurring events like traffic jams during peak hours. Therefore, they try to avoid driving during those times, or use alternative routes with less traffic. However, if the scenario contains unexpected events that the agents cannot foresee, e.g., accidents or heavy weather conditions, using an iterative approach is not an appropriate choice. First, a user equilibrium will not be reached in such a scenario because agents do not have enough information to choose optimal routes and daily activity plans. Another problem is the optimization process itself. Even if an agent chooses its routes randomly due to a lack of information, it will eventually find a good route if it tries enough different routes.

Figure 30.1 shows a simple example scenario where an iterative approach would produce illogical and faulty results. In Figure 30.1(a), an agent's planned route in a sample network is shown, including the times when the driver passes each node of the route. Clearly, those times are only valid if no exceptional event occurs. Figure 30.1(b) shows a link where an event, like an accident, blocks that link for two hours. As a result, the agent reaches its destination two hours later than expected (Figure 30.1(c)). When this scenario is iterated, the agent recognizes that its route has a much higher travel time than expected and therefore it will choose another route. The traffic jam caused by the accident will probably also increase travel times on links next to the blocked link. Therefore, the agent might find a route which is quite different than the original one (Figure 30.1(d)). A closer look at the node where the new route deviates for the first time from the original one shows that this occurs even before the accident happened, which is unfeasible and illogical.

An obvious solution to avoiding such problems is using an alternative simulation approach without an iterative optimization process. The next section discusses such an approach and the requirements that must be fulfilled.

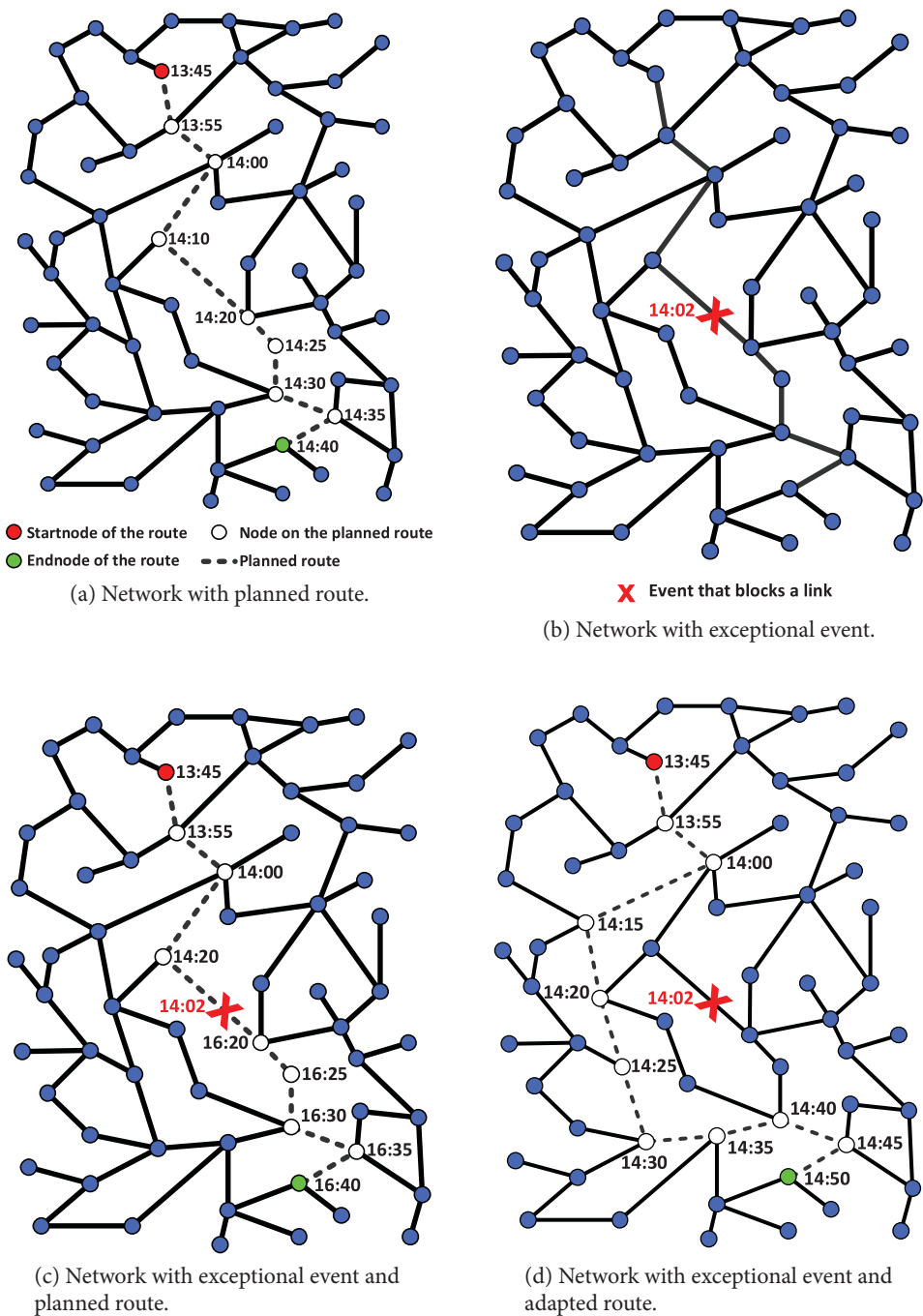


Figure 30.1: Exceptional event in a network.

30.3.2 Within-Day Replanning Approach

A within-day replanning approach uses a significantly different strategy from that of an iterative approach. Instead of multiple iterations, only a single one is simulated. Thus, it is now essential that agents can adapt their plans during this iteration without having information from previous

iterations available. To do so, they have to continuously collect information and take into account their desires, beliefs and intentions when they decide how to (re)act.

While iterative approaches can use best-response modules, a within-day approach has to use something that might be called a best-guess module. Travel times are an obvious example. In an iterative approach, travel times can be collected from the previous iteration or even be averaged over several past iterations. The nearer a stable system is to a relaxed state, the smaller the differences in travel times between two iterations. This is not possible in a within-day approach. Even if an agent has perfect knowledge, it can only assume how the traffic flows will evolve in the future. To do so, it can take different information into account to estimate travel times. It could, for example, take travel times from a typical day without exceptional events and combine them with information it gathers during the simulated day. Depending on the amount and the quality of this information, the agent might rely more or less on its experience.

Therefore, the decision-making process of an agent becomes an important topic. In an iterative approach, each agent has total information and can thus select the best route. Due to limited available information, this is not possible in a within-day approach. One agent could, for example, choose a route where expected travel time is very short, but also very uncertain. Another agent might not be willing to take that risk and therefore select a longer route where the assumed travel time is more reliable. Perception of information might also vary between agents; one could rely on media traffic information, another might ignore it.

Each within-day replanning action is categorized by two parameters—the replanned element of the plan (an activity or a trip) and the point in time when the replanned plan element is executed (right now or at a future point in time). If an activity is replanned, several changes are possible. Its start and end time can be adapted, its location can be changed, it can be dropped, or created new from scratch. For a trip, origin and destination, route, mode of transport and departure time can be replanned. Often replanning one single plan element results in a chain reaction that forces replanning of other plan elements. If, for example, an activity is dropped, the trips from and to this activity have to be merged.

The second parameter categorizing a replanning action depends on when the replanned plan element is executed. This could be either the currently performed plan element or one being performed in the future. Clearly, in a currently performed plan element, not all previously mentioned replanning actions could be conducted, e.g., start time of an activity or transport mode of a trip currently being performed can no longer be adapted.

Due to the limited available information, a within-day replanning approach will, in contrast to an iterative approach, not converge to a user equilibrium. Decisions made during the simulated time period may seem to be optimal when they are made. However, evaluated retrospectively, an agent might realize that they were not.

Figure 30.2 shows how within-day replanning can be integrated into MATSim's iterative optimization loop. An additional block builds another (inner) loop with the mobility simulation. Depending on the type of simulated scenario, the outer loop can be skipped.

30.3.3 Combined Approaches

An alternative to iterative, or within-day replanning only approaches, is to combine them. An obvious application is solving situations that cannot be planned exactly in advance, like parking or car sharing. An agent is, for example, able to plan a parking activity, but it cannot anticipate which parking spots will be available when the agent arrives. Thus, within-day replanning can be used when the agent starts its parking choice.

Other agents might want to share their cars, so an actual meeting must be confirmed. This can be ensured using within-day replanning. If the driver arrives too early, a *waiting* activity is added to its plan; otherwise the agent being picked up will perform a *waiting* activity until the car arrives.

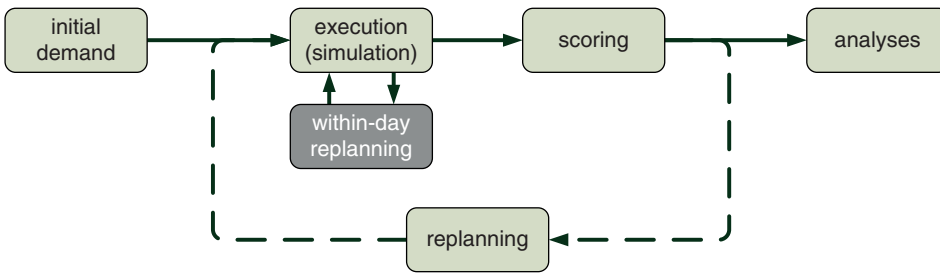


Figure 30.2: (Iterative) within-day replanning MATSim loop.

30.4 Implementation

30.4.1 General Thoughts

Within-day or en-route replanning means that travelers replan during the day or while they are on their route. This means that the simulation needs to find some way to influence the agent while the mobsim (network loading) is running. For the MATSim main network loading module, the so-called QSim, this could be achieved by inserting an agent-loop, as follows:

```

void doSimStep() {
    for ( each agent ) { // <-- agent loop
        agent.doSimStep() ;
    }
    for ( each link ) {
        link.doSimStep() ;
    }
    for ( each node ) {
        node.doSimStep() ;
    }
}

```

In this loop, each agent has the chance to deliberate in every time step. Clearly, the agent can decide that he/she has nothing to deliberate and return immediately.

Such an approach does, however, lead to computational challenges. Going through all links and nodes in every time step is already an expensive operation and a number of efficiency improvements (such as “switching off non-active links”) are contained in the code. Also, the number of links or nodes is typically an order of magnitude smaller than the number of synthetic persons in a scenario. Thus, some massive optimization would have to be undertaken in order to make the above approach computationally efficient.

An alternative approach to the above is to ask each agent only when a decision needs to be made. The most important decision for a driver is to choose the next link, i.e.,

```

class MyDriverAgent implements DriverAgent {
    ...
    @Override
    public Id<Link> chooseNextLink() {
        <algorithm to determine ID of next link>
        return nextLinkId ;
    }
}

```

Similar implementations are needed for all other queries that could be asked of the agent, for example:

- Should the trip end on the current link?
- Should the agent get off at the current stop?
- What is the ID of the vehicle to be used for a trip?

From the agent's perspective, such an approach might be called *event driven*, since the agent performs only mental activity at such events.

There is, indeed, a mechanism to program such agents and to insert them into the QSim. This is discussed in more detail in Section 30.4.3.

A challenge inherent in that approach is that the complete agent needs to be re-programmed. This agent needs to have enough capabilities to be oriented about itself; for example, it needs to be able to compute plausible routes.

On the other hand, there are situations where the capability to decide the turn at each intersection while en-route is, in fact, not needed. For example, for typical evacuation applications, it makes sense to start all agents on their normal daily plans. When an emergency warning is distributed, the simulation can go once through all agents and decide how they react. This will be done by replacing some, or all, future elements of the current plan. In some applications, this may happen more than once; for example, if recommended evacuation directions change because of a shift in the wind. In other applications, evacuating agents could become stuck in unexpected congestion which might trigger en-route re-routing. This may, however, be restricted to relatively small regions, and it may be sufficient to go through such a replanning loop, perhaps every 300 simulated seconds.

For such applications, the plan-based approach (Section 30.4.2) is more suitable. Rather than having each agent answering certain queries in every time step or at every intersection, the plan-based approach first waits for a trigger (such as an emergency warning, or unexpected congestion), then decides on the affected agents, then goes through those agents and changes the future part of their plans. This is not only conceptually easier than having every agent answer for him-/herself, but it is also computationally more efficient, since it is only called when it is triggered and impacts only the affected agents.

Overall, implementers and users will have to balance their needs. If there are relatively few times when agents should re-plan, and these times can be easily identified by, i.e., corresponding to an emergency signal, then this is an indicator for the plan-based approach. If, on the other hand, an agent goes into the simulation mostly or entirely without a plan, like an entirely reactive taxi driver, then this speaks for replacing the agent.

MATSim provides infrastructure for both approaches. The plan-based approach currently provides more support infrastructure, i.e., many important use cases can be implemented by re-using existing methods. The approach that replaces the agent, in contrast, provides more flexibility. In particular, it allows agents to make decisions at the latest possible time without additional computational overhead. While this is not entirely realistic behaviorally, such an approach is often desirable from a simulation perspective, where one does not want reproducibility of simulations depend on, e.g., random elements such as how far an agent plans ahead.

30.4.2 Implementation Alternative 1: Plan-Based Implementation

When adding within-day replanning to MATSim, its iterative loop (see Figure 1.1) has to be adapted as shown in Figure 30.2. On one hand, the additional *within-day replanning* module is added, which interacts with the mobsim. On the other hand, multiple iterations are only necessary if a combined simulation approach is used.

The implementation is realized as so-called *MobsimEngine* which can be plugged into the *QSim*. In every simulated time step, the *QSim* iterates over all registered *MobsimEngines* and allows them to simulate the current time step. Besides simulation of the traffic flows, those engines are also able to let agents start or end activities. The engine containing the within-day replanning logic (called *WithinDayEngine*) does not simulate traffic flows, but tracks agents and adapts their plans. Doing so is separated into two steps. First, agents whose plans have to be adapted in the current time step are identified. In a second step, the adaption of their plans is performed.

Figure 30.3 shows the structure of the *WithinDayEngine*. Multiple *Replanners* can be registered to the engine. Each *Replanner* represents a unique replanning strategy like re-routing or time mutation and uses a set of *AgentSelectors* that communicate with agents and select those who are given the opportunity to adapt their plans. An *AgentSelector* can be seen as an information-distributing unit, like a radio station or a policeman. Therefore, not every *AgentSelector* communicates with all agents. For example, agents at home will probably listen to the radio, but agents walking in the park will not. Each *AgentSelector* returns a list of agents to its superior *Replanner*, which then adapts those agents' plans.

Responsibilities are divided between *Replanners* and *AgentSelectors*. The first ones are responsible for adapting the agents' plans, but they should not check whether an agent should be replanned or not. If, for example, a *Replanner* updates an agent's route, it has to be ensured by the *AgentSelectors* that only agents who are currently performing a leg are replanned. In turn, *AgentSelectors* should select agents who have to be replanned but should not change their plans. As a result of this division, the often time-consuming replanning of the agents' plans can be performed using parallel threads, which leads to an almost linear speed-up. In general, simulation results do not depend on the order in which agents are replanned. *Replanners* which use random

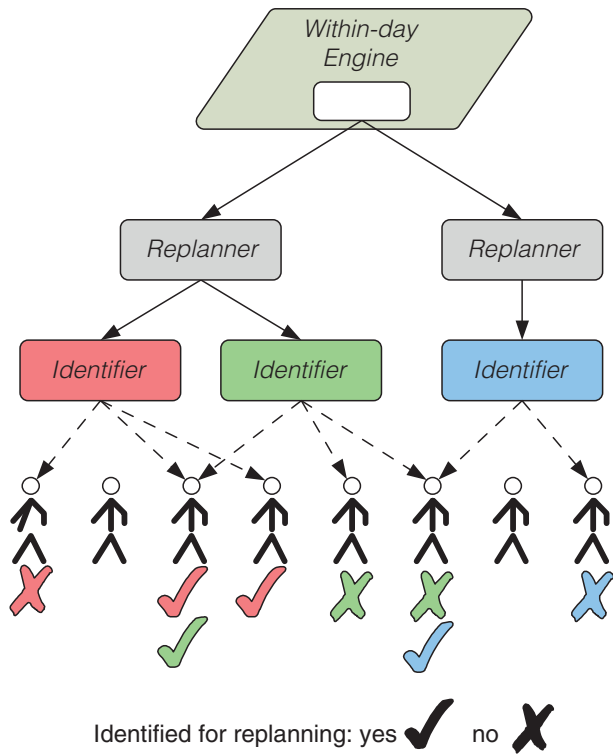


Figure 30.3: *WithinDayEngine*.

numbers are a special case. In the present implementation, their *random number generator* is re-initialized for every replanned agent, using a deterministic value (e.g., a combination of the agent's ID and the current time step). On one hand, this ensures that an agent's decisions can be reproduced even when the global sequence of random numbers changes. On the other hand, the simulation outcomes do not change if the number of threads used for the replanning is changed.

Running the `AgentSelector(s)` to select those agents who have to adapt their plans is performed sequentially. On one hand, an `AgentSelector`'s runtime is typically very short and therefore no significant performance losses are expected. On the other hand, this makes the design robust so it cannot produce race conditions which could occur if multiple instances of an `AgentSelector` run concurrently. An example would be an `AgentSelector`, which selects agents on household level, i.e., if a member of a household is identified, also all other members are added to the list of agents who have to be replanned. In an approach with parallel running instances of an `AgentSelector`, an instance could identify member "A" of a household while concurrently another instance could identify member "B" of the same household. As a result, the household's members would be duplicated in the list of agents to be replanned—once added by each `AgentSelector` instance.

Replanner implementations are available for any basic change of an agent's scheduled daily plan. All trips and activities can be adapted, although some replanning operations are not available when trip or activity has already been started. Possible adaptations are:

- current trip (route, destination),
- future trip (add, remove, mode, route, origin, destination),
- current activity (end time), or
- future activity (add, remove, location, type, start and end time).

For complex plan adaptations, those basic Replanners can be combined. If, for example, an agent currently performing a trip changes the destination of its next activity, routes of the current and next trip must be adapted.

Additionally, four basic `AgentSelectors` have been implemented so far. They identify agents, which are...

- performing an activity,
- performing an activity which will end in the current time step,
- performing a trip, or
- performing a trip and are going to move to another link.

Often, only a subset of the population, e.g., only male agents, or agents currently traveling in a car, needs to be identified. To prevent that the same functionality having to be implemented multiple times, so-called `AgentFilters` are introduced. Their task is to remove agents not meeting the filter criteria from an agent set. Using `AgentFilters` not only avoids duplicated code, but can also reduce computation effort: for example, two `AgentSelectors` which should identify only agents currently traveling in a certain part of the network. Without `AgentFilters`, each of them would have to track all traveling agents and their current positions. When this functionality is moved to an `AgentFilter`, the two `AgentSelectors` can share a single instance of that filter.

Basically, simple and re-usable functionality should be implemented as `AgentFilters`, while more complex and/or decision-making functionality should be part of an `AgentSelector`. Again, an example: e.g., a scenario modeling the search for a parking space: a filter can be utilized to take only agents currently traveling by car into account. The `AgentSelector` solves the more complex tasks, such as deciding when the agent starts its search, or selecting the searching strategy to be applied.

Three basic AgentFilters have been implemented so far. They filter agents which are not...

- part of a predefined agent set,
- currently using a transport mode included in a given set, or
- currently located on a link included in a predefined set.

In addition to the logic identifying agents and adapting their plans, another important within-day replanning framework component is code that continuously collects information and provides it to the AgentSelectors. These decide, based on that data, whether agents are replanned or not. In a time step-based approach—as realized by the QSim—collecting, analyzing and aggregating data, as well as providing it, can be easily realized. Figure 30.4 shows the structure of a QSim’s time step. Each time step is separated into three phases:

Phase 1:

```
before time step
```

Phase 2:

```
do sim step
```

Phase 3:

```
after time step
```

During phase 2 all registered MobsimEngines simulate the current time step. Phases 1 and 3 allow code execution before or after simulation of the current time step. A class can collect data such as link travel times during phase 2, phase. Then, the collected data can be analyzed and aggregated in phase 3. In the next time step, the WithinDayEngine’s AgentSelectors can use that data for their decisions. The WithinDayEngine is always the first MobsimEngine executing its doSimStep method, ensuring that no agent has changed its status since phase 3 of the previous time step. As a result, the AgentSelectors make their decisions on current data.

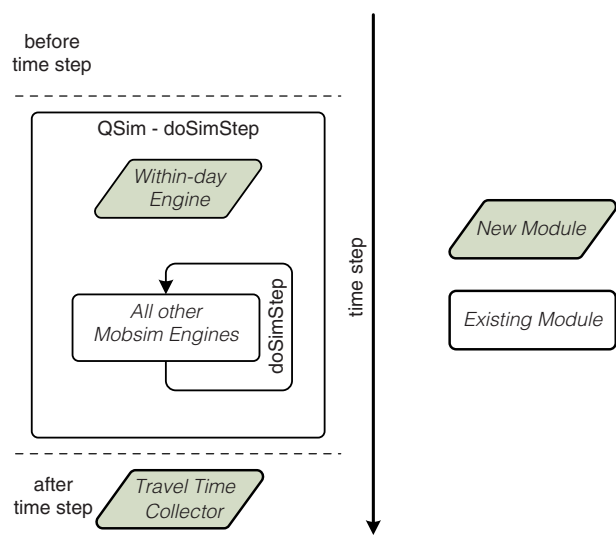


Figure 30.4: QSim time step.

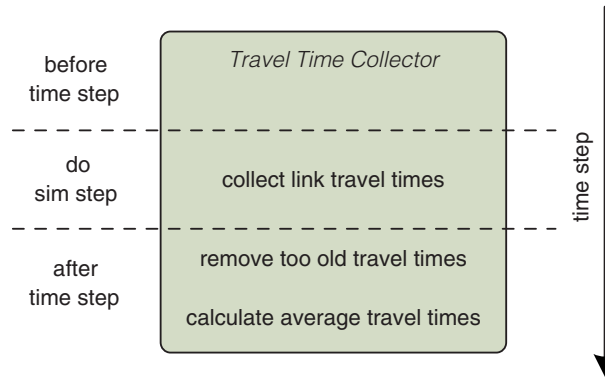


Figure 30.5: TravelTimeCollector.

An example of this type of class is the so-called *TravelTimeCollector*. It provides actual link travel times to the Replanners by collecting and averaging travel times of agents that have recently passed a link during a given time. A typical time span is 15 minutes; older link travel times are ignored. Specific time span duration has an important impact on travel times reported to the Replanners. On one hand, significant changes in link travel times will be communicated very slowly, if the time span is too long. On the other hand, a too short duration will overrate outliers.

The *TravelTimeCollector* is a simple, but efficient, implementation of a within-day travel time calculator. It does not incorporate features like traffic flow predictions or dynamic recent travel times weighting based on historic data. Because it does not factor in such features, it is very robust, even in scenarios where traffic flow conditions change dramatically.

The current MATSim code differentiates between *Person* and *MobsimAgent*. *Person* can be seen as a very simple Q-learning entity, possessing multiple *Plans* (“actions”), each with an expected score updated with every plan run. Thus, a *Person* is consistent over the iterations; in fact, the internal state of each *Person* is written to file at the end of the iterations. *MobsimAgent*, in contrast, is instantiated every time the *QSim* is called, and does not exist beyond the *QSim* running time. A *MobsimAgent* is essentially reactive, queried by the framework about decisions when approaching intersections, arrival points, or public transit stops. In the standard implementation, these queries are answered by the plan, but other implementations can be used and/or additional *MobsimAgents* can be added which do not correspond to *Persons*.

This leads to a question; should within-day adaptations to the *Plan* be passed through to the *Person*? Let us call the actual trajectory through the system the “executed plan”. This can be different from the original plan, i.e., a different route, different departure times, different modes, etc. The original plan cannot just be replaced by the executed plan, since it is not clear that the executed plan, when used as input, will have itself as expected output. In consequence, it is not possible to treat the executed plan together with the just-obtained score as an action-value pair in the sense of Q-learning, since the score was obtained from the *original* plan, not from the executed plan.

As a result, the code uses a copy of the original plan and modifies the copy. The score, however, is given to the original plan. The implementation is able to *also* memorize the executed plan and add it to the set of plans. This functionality, however, is experimental.

In certain situations, setting the original to the executed plan clearly does not make sense; a parking search is one (Waraich et al., 2013c, 2012).

A person’s plan contains, as destination, the location where a free parking space is expected. However, if the agent realizes in the mobility simulation that there is no free space left, it starts looking for a free parking spot. As a result, the agent’s route is extended. This extension has to

be local in the agent's route, since it is only necessary in the current iteration and probably not in another one, where the initially selected parking spot is available.

Capabilities of this within-day replanning implementation are shown and discussed by Dobler (2013), based on two sets of experiments. The first set is based on a model of Zürich city, where it is assumed that capacities of several city-center arterial roads are drastically reduced during the morning peak. Traveling agents are given the opportunity to bypass the resulting traffic jams by adapting their routes, using within-day replanning. As a result, average travel time of an agent affected by the incident is reduced from 42 to 23 minutes. Also interesting is that even if only 50 % of the population adapts its routes, average travel times are reduced to 25 minutes.

The second set of experiments uses within-day replanning to create agents' initial routes. The results are compared to runs where routes are created before the simulation starts, without traffic flow information. Results indicate that agents' average travel times are already very close to the values in a relaxed state. When using MATSim's traditional approach, 10 to 15 iterations must be performed before average travel times reach this level.

30.4.3 Implementation Alternative 2: Replacing the Agent

According to Russel and Norvig (2010), an agent is “anything that can be viewed as perceiving its environment through sensors and acting upon that environment through effectors.” As stated above, MATSim has agents on two levels:

- Person is a Q-learning agent that is persistent over the iterations.
- MobsimAgent is a reactive agent that only exists during the mobsim.

For the Q-learning agent, perception works through the events; i.e., events are used to compute the score, build mental models to generate alternatives, etc. Acting on the environment works through plan selection.

For the reactive agent, perception works more directly through callback methods, such as the simulation notifying the agent it has just moved through an intersection. Acting on the environment works through making decisions at decision points, e.g., about turning directions at intersections, or whether to board a certain bus.

As discussed, the approach described in Section 30.4.2 assumes that the reactive agent still has followed (and generally follows) a plan. There may, however, be situations where this is inappropriate: for example, when the agent makes up the route as it goes, or when one wants to investigate models where each agent has its own perception and deliberation, rather than some external algorithm modifying its plan. As also mentioned earlier, there is no clear rule governing when and where an approach is better; it depends both on both project requirements and on the developer's personal preferences. Here, with this in mind, we will look at MobsimAgents that no longer have a pre-computed plan, but make decisions as they go. There is also a class DynAgent, which wraps around MobsimAgent, making it easier to use and providing additional infrastructure (Section 23.4).

30.4.3.1 Agent Interface

The DriverAgent interface structurally looks as follows:

- `Id chooseNextLinkId()`—agent is asked at intersections and needs to return how to proceed.
- `boolean isWantingToArriveOnCurrentLinkId()`—agent is asked if it wants to arrive on the current link.
- `void notifyMoveOverNode(Id newLinkId)`—agent is notified that it has traversed the intersection and entered a new link.

The rest comprises relatively simple bookkeeping methods like `getId()`—the agent needs to know its own identifier.

If it is assumed that the agent does not only replan en-route, but also while at activities, then the `MobsimAgent` interface also must be implemented. This is a bit more involved; important methods are:

- `endLegAndComputeNextState(...)`—agent is notified that the current transport leg has ended, and the agent internally needs to decide how to continue.
- `endActivityAndComputeNextState(...)`—agent is notified that current activity has ended; the agent internally needs to decide how to continue.
- `setStateToAbort(...)`—if a leg or an activity was not ended cleanly: this could happen if `chooseNextLinkId()` returns a link that is not outgoing from the current node.¹
- `getState()`—agent needs to return its current state, which essentially either returns `ACTIVITY` or `LEG`; most important here is that the framework obtains information about whether the agent wants to start a new activity or leg.

Again, everything else concerns bookkeeping methods.

30.4.3.2 Agent Insertion

The code accepts several ways to insert such a self-programmed `MobsimAgent` into the code, but the preferred method is using the `AgentSource` interface, as follows:²

```
class MyAgentSource implements AgentSource {
    // constructor
    MyAgentSource ( Guidance guidance ) {
        ...
    }
    public void insertAgentsIntoMobsim() {
        // insert agent:
        MobsimAgent ag = new MyMobsimAgent( guidance ) ;
        qsim.insertAgentIntoMobsim(ag) ;

        // insert vehicle:
        // ...
        qsim.createAndParkVehicleOnLink(veh, linkId );
    }
}
```

Guidance helps the agent with making decisions, see below.

30.4.3.3 Perception, Decision, Integration

The agents somehow need to perceive their environment. The simulation tells the agent where it is, via `notifyMoveOverNode(Id<Link> nextLinkId)`. In general, however, this will not be sufficient. For example, the agent may want to be informed about congestion, or evacuation directions.

A general way to achieve this is to use the `Events` channel.

We would probably suggest separating observer, guidance, and the agent itself.

¹ Despite the name of the method, the agent can recover.

² See <http://matsim.org/javadoc> → main distribution → the `AgentSource` class for a pointer to a working code example.

Observer The observer would probably listen to events:

```
class MyObserver implements BasicEventHandler {
    @Override
    public void handleEvent(Event event) {
        ... // memorize information
    }
    ...
}
```

For working code, see <http://matsim.org/javadoc> → main distribution → `RunOwnMobsimAgentWithPerception` class and related.

Guidance A guidance object might give advice to agents. It could, for example, be designed as follows:

```
class MyGuidance {
    MyGuidance( MyObserver observer ) {
        ...
    }
    Id<Link> chooseNextLinkId( Id<Link> currentLinkId ) {
        ... // compute and return decision
    }
}
```

For working code, see <http://matsim.org/javadoc> → main distribution → `RunOwnMobsimAgentWithPerception` class and related.

Agent The agent needs access to the guidance object:

```
class MyAgent implements MobsimDriverAgent {
    MyGuidance guidance ;
    MyAgent( MyGuidance guidance ) {
        this.guidance = guidance ;
    }
    ...
    @Override
    Id<Link> chooseNextLinkId() {
        return this.guidance.chooseNextLinkId( this.currentLinkId ) ;
    }
    ...
}
```

For working code, see <http://matsim.org/javadoc> → main distribution → `RunOwnMobsimAgentWithPerception` class and related.

Control script This would be plugged together by a variant of the following script:

```
Controller ctrl = ... ;
...
// create observer object:
MyObserver observer = new MyObserver() ;
// add into events channel:
ctrl.addEventsHandler(observer) ;
// create guidance object:
MyGuidance guidance = new MyGuidance( observer ) ;
// create mobsim factory and set into controller:
ctrl.setMobsimFactory(new MobsimFactory(){
    public Mobsim createMobsim(Scenario sc, EventsManager ev ) {
        MobsimFactory factory = new QSimFactory() ;
```

```

    QSim qsim = (QSim) factory.createMobsim(sc, ev) ;
    // add agent source into mobsim:
    qsim.addAgentSource( new MyAgentSource( guidance ) ) ;
    return qsim ;
}
}) ;
...
ctrl.run() ;

```

The above “script” uses an anonymous class for the MobsimFactory. This method of writing code is quite convenient for adapting MATSim to individual needs, also see Chapter 45.

For working code, see <http://matsim.org/javadoc> → main distribution → RunOwnMobsimAgentUsingRouter class and related.

30.4.3.4 DynAgent

As stated earlier, there is also a class DynAgent. It wraps around MobsimAgent, making it easier to use and providing additional infrastructure (Section 23.4).

CHAPTER 31

Making MATSim Agents Smarter with the Belief-Desire-Intention Framework

Lin Padgham and Dhirendra Singh

31.1 Basic Information

Entry point to documentation:

<http://matsim.org/extensions> → bdiintegration

Invoking the module:

See <http://matsim.org/extensions> → bdiintegration

Selected publications:

Padgham et al. (2014)

31.2 Introduction

In this chapter, we introduce a MATSim extension allowing a developer to program (some of) an agent's decision-making in a BDI (Belief Desire Intention) system, while actual actions and environment percepts occur within MATSim.¹ This allows sophisticated modeling of agents within a BDI framework, using the concepts of goals, hierarchical abstract plans (containing sub-goals)

¹ This work was supported by the ARC Discovery DP1093290, ARC Linkage LP130100008 and Telematics Trust grants. We would like to thank Agent Oriented Software for use of the JACK BDI platform and Kai Nagel, Todd Mason, Sewwandi Perera, Edmund Kemsley, Oscar Francis, Daniel Kidney, Andreas Suekto, Qingyu Chen, and Arie Wilsher for their contribution to the BDI platform integration framework and to these applications.

How to cite this book chapter:

Padgham, L and Singh, D. 2016. Making MATSim Agents Smarter with the Belief-Desire-Intention Framework. In: Horni, A, Nagel, K and Axhausen, K W. (eds.) *The Multi-Agent Transport Simulation MATSim*, Pp. 201–210. London: Ubiquity Press. DOI: <http://dx.doi.org/10.5334/baw.31>. License: CC-BY 4.0

and percepts (information from the environment), as well as information about the current situation. For example, we used it to model residents in a bushfire² evacuation, as well as an incident controller in an evacuation scenario. The residents may receive information about the bushfire from the fire simulation, as well as warnings and messages from the incident controller agent. They may well have to pick up children, check on neighbors and communicate with other family members, etc. Their plans enable decision-making, which will result in actions executed within MATSim.

In standard MATSim usage, intelligence within individual agents' behavior arises from co-evolutionary algorithms in the replanning phase. This is based on agents evaluating—via a scoring function—the plan they have executed during a given day and modifying this to obtain a new plan, until all agents have acceptable plans; the system then reaches a stable state. This approach, however, only works for applications where one can assume that the agents adjust and refine their behavior over many iterations, to eventually obtain their standard *modus operandi*. For applications such as emergency management, agents must react immediately to the situation as it evolves, doing so in an “intelligent” manner.

The chapter on Within-Day Replanning introduces two approaches to the mobsim component which address the need to be more reactive to an evolving situation. The first allows a centralized MATSim process to identify sets of agents that should have their plans modified, then runs one or more processes to adjust agents' plans. The second rewrites the agent, so that instead of following a specified plan, the agent invokes a decision-making process at all possible decision points. By integrating a BDI agent platform with MATSim (Padgham et al., 2014), we allow autonomous individual decision making to be programmed in specialized and powerful systems developed specifically for this purpose, balancing reactive behavior and goal-based commitment. Different BDI platforms have different strengths, but are, in general, based on a simplified psychological/philosophical view of how people behave, facilitating a high level specification of complex human behavior. These systems have been demonstrated to be very efficient for building complex applications (Benfield et al., 2006). Provided the appropriate system interface support is developed, any BDI system can be coupled to MATSim, as described here. Until now, we have used three different BDI systems, for which the system level interface is available. The decisions made in the BDI system are then inserted into the relevant agents' MATSim plans, allowing the MATSim agents to operate in the same efficient manner as in standard MATSim.

31.3 Software Structure

Our framework supports independent execution of MATSim and the BDI platform, with synchronization via the infrastructure provided. They can either run within a single process (in separate synchronized threads, or sequentially in a single thread), or in two separate processes (synchronizing using inter-process communication, such as sockets). The former is, of course, considerably more efficient. Conceptually, for every MATSim agent whose decision making is to be carried out in the BDI system, a BDI agent must be created. The BDI counterpart can be regarded as “the brain” associated with the MATSim agent. It is possible to have BDI agents with no MATSim counterpart and vice versa. For example, in our bushfire application, the incident controller has no MATSim agent, as he does not move on the road network. He receives information about the fire and has some static location information; his role in the simulation is to issue warnings and evacuation advisories, which, in turn, affect the resident agents. There may also be MATSim agents that do not have a BDI counterpart. For example, in a taxi modeling application, there may be MATSim

² Bushfire is the Australian term for what is otherwise known as a wildfire or forest fire.

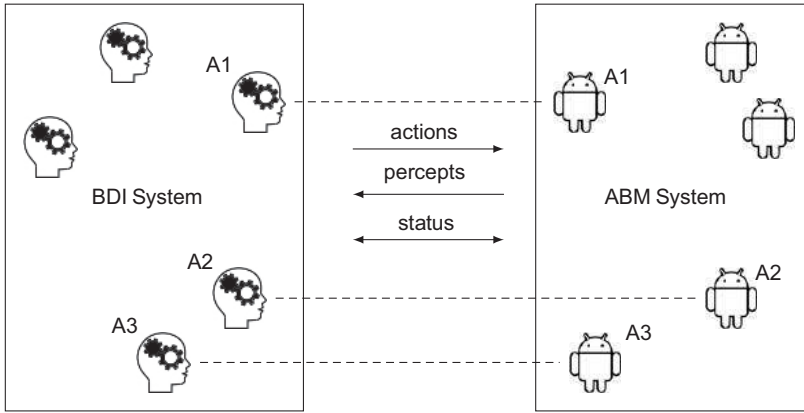


Figure 31.1: Conceptual BDI-ABMS integration architecture.

Source: Figure adapted from Padgham et al. (2014, Figure 1) distributed under the Creative Commons Attribution Non-Commercial License

agents using the road network, but with no need for complex decision-making modeling; these may exist only within MATSim.

Figure 31.1 shows the two parallel systems’ basic architecture and the information passed between them at each time step.

The structure of the data components passed between the MATSim agent and its BDI counterpart is shown in Table 31.1 and consists of *BDI Actions*³, *Percepts* and *Queries*. As indicated in Figure 31.1, BDI-actions are always initiated by the BDI system. Their status field, however, can be modified by both systems. When a BDI action such as *DriveTo(loc)* is decided by the BDI agent, the BDI system sets the status of this action as “INITIATED”. MATSim will then set its status to “RUNNING”, which will probably remain in this state for several steps. When the *loc* destination is reached, the MATSim routine will set the status to “PASSED” and the BDI system will continue reasoning about the next stage of agent behavior. If desired, the MATSim routine can also detect situations which should be conveyed as “FAILED” and pass this to the BDI counterpart. For example, if there is a BDI action to meet at a location and time and the MATSim agent is delayed in traffic, the BDI action implementation in MATSim can be programmed to detect the missed deadline and set the status to “FAILED”, at which point the BDI agent will attempt failure recovery (as part of the BDI infrastructure). The BDI system can also set the status to “ABORTED”—for example, if information arrives requiring a different action—in which case, it is canceled within MATSim. The BDI system can also set status to “SUSPENDED”, though this is not currently implemented.

To manage BDI actions, we provide a *MatsimAgentManager* class responsible for updating BDI actions status for all agents. At each step, the *MatsimAgentManager.updateActions(...)* function identifies (from the information package supplied by the BDI system) all agents initiating, aborting, or suspending actions. These are the agents which may require their MATSim plans to be modified. For each agent that has some action with a status “INITIATED”, the action is passed to the agent’s action handler class *MatsimActionHandler* via a call to *MatsimActionHandler.processAction(agentID, actionID, params)*. This function, based on the action, calls an appropriate helper function that performs required modifications to the MATSim plan and other relevant bookkeeping, to ensure that success and failure are observed (via

³ We call these actions BDI Actions to distinguish them from actions in the ABMS (Agent-Based Modeling and Simulation) which may include lower level or additional actions.

Components of The Data Package Provided to Specific Agents Via The Interface:

Component Type	Component fields
BDI action	<i>< instance_id, action_type, parameters, status ></i>
Percept	<i>< percept_type, parameters, value ></i> (parameters and value may be complex objects)
Query	<i>< query, response ></i>

BDI Action Status:

State	Description
INITIATED	Initiated by BDI agent and to be executed
RUNNING	Being executed, set by the simulation agent
PASSED	Completion detected and set by the simulation agent
FAILED	Failure condition detected and set by the simulation agent
DROPPED	Aborted by the BDI agent
SUSPENDED	Temporarily suspended by the BDI agent

Table 31.1: Data Passed Between The BDI and ABMS Systems

appropriate MATSim callbacks) and that status is reported back to the BDI system. For example, for a `DriveTo` action, a `processDriveTo(agentID, loc)` function is executed to determine the leg associated with `loc`, obtain a route using the MATSim router and insert this into the MATSim agent's plan. The standard MATSim execution then follows this plan at each subsequent step. If the `processAction` function returns a success status indicating that the action was handled successfully, then `updateActions` changes the status for this action to "RUNNING"; otherwise, it sets it to "FAILED."

Sometimes, a running action can also fail in the ABMS for some reason. For instance, a `DriveTo` (`loc`) action could fail due to a road-closure in a bushfire evacuation simulation. While this functionality is supported by our infrastructure, it has not yet been used in the applications we have built with MATSim. Failing actions will soon be added for some applications. Aborting and suspending are also not currently implemented for MATSim. This would be accomplished by having appropriate functions declared which reset the plan contents of the agent to a 'holding state' (activity with infinite end time), maintaining the removed contents of a suspended plan in some data structure for eventual resumption.

Percepts capture information identified as necessary for the BDI agent's reasoning. Typically, this is any information leading to triggering of a BDI-goal, or causing an executing goal/plan to be re-evaluated. Approaching a destination is one example. MATSim callbacks are used to capture the relevant information within MATSim; this is then provided to the BDI counterpart via our infrastructure. The appropriate MATSim event is caught with `AgentActivityEventHandler.handleEvent(event-type)`. The `handleEvent(event-type)` function then first checks whether the agent receiving the event is one registered for a percept that triggers with this event type, and if so, calls the appropriate function to calculate the percept's value and add it to the percept container for that agent, to be sent to the BDI system. Termination conditions (PASSED and FAILED) of BDI actions are also similarly detected.

Instead of passing back the percept in these cases, the relevant action and its status is edited and passed back. For example, a BDI action `DriveTo(loc)` should succeed when the agent reaches the link closest to this location. To achieve this, we implement `handleEvent(PersonArrivalEvent)`, which will then trigger for every agent arriving anywhere. If the agent has a current (`DriveTo`) BDI action being monitored, then `arrivedAtDest(agentID, loc)` is called to ascertain whether the

`PersonArrivalEvent` caught does match the link closest to the coordinates of the desired destination. If it does, the action status of that `DriveTo` action for that agent is changed to `PASSED` and the action is removed from the monitoring list.

This approach conveniently uses MATSim callback infrastructure. However, we note that it will generate an event that must be processed any time any agent arrives anywhere, although most will not be an arrival at a desired destination. This is a substantial overhead; we may eventually consider collecting (some) percepts and state information for determining action status, in a separate, more efficient global processing at the end of the step.

Queries are defined for any information that the BDI system may want to request from MATSim during its reasoning process. Typically, queries are based on plans' context conditions, which must be evaluated to determine if a plan is applicable. Each query structure must be defined and the code must be supplied on the MATSim side to call the relevant functions to provide the response. Similar to the `MatsimActionHandler` class, we have a `MATSimPerceptQueryHandler` class containing a `queryPercept(agent, query, response)` function. This function then uses the query string received to extract the percept type and make a specific function call to obtain and provide the results. For example, if an agent `agentID` sends a `queryPercept(agentID, 'RequestLocation agentX', loc)` query to request the location `loc` of some agent `agentX` (possibly itself), then the `queryPercept` function will execute the clause:

```
if percept_type = "RequestLocation"
    loc = getLocation("agentX")
```

The `agentID` of the requesting agent, obtained from the data package, is always provided to the query response function, in case it is required, although in this case it is not. Queries can be made at any point during the BDI execution and are answered immediately. They have no effect on the MATSim simulation.

A number of commonly used BDI actions and percepts are defined as part of our integration infrastructure. New ones can be added as part of developing a specific application, as described in Section 31.4. This structure allows all high-level decision making to be carried out by individual agents, within the BDI-system, which is designed and optimized for this purpose with regard to both representation and execution. On the MATSim side, specified functions simply modify the agents' MATSim plans (in parallel, if desired), retaining the standard MATSim simulation execution where each agent just follows its MATSim plan. This approach allows for both simplicity and efficiency at the lower level.

31.4 Building an Application Using BDI Agents

We focus here only on what must be done to integrate BDI agent reasoning into MATSim. To learn about BDI design and development, we refer the reader to Padgham and Winikoff (2004), as well as the excellent "practicals" (tutorials) available as part of the JACK platform⁴. In Figure 31.2, we show part of a taxi agent design, in an application involving taxis operating within MATSim. Here, the percept `ClosetoDest` (potentially) triggers a plan `GrabJob`. Plans have context conditions which indicate whether or not they are viable in the current situation, as a response to a percept, or a way of achieving a goal. Let us assume, in this example, that the plan `GrabJob` has the context condition $(\text{Location}(\text{self}, \text{loc})) \wedge \text{board.job.loc} \wedge (\text{distance}(\text{board.job.loc}, \text{loc}) < 4\text{km})$. Thus, the figure at the left of the diagram can be understood as the rule:

$$\text{ClosetoDest} \wedge \text{Location}(\text{self}, \text{loc}) \wedge \text{board.job.loc} \wedge (\text{distance}(\text{board.job.loc}, \text{loc}) < 4\text{km}) \rightarrow \text{GrabJob}$$

⁴ <http://aosgrp.com/products/jack/>

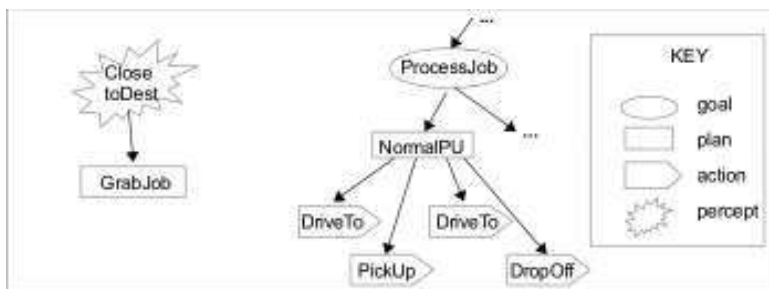


Figure 31.2: Excerpt of taxi design.

There are two pieces of information in this rule that must come from MATSim: first, the agent is close to its destination (*ClosetoDest*) and second, the agent's current location (*Location(self, loc)*). We could have MATSim send the agent location at every step. However, this is unnecessary overhead; instead, we send *ClosetoDest* as a percept. This requires the BDI agent to query its location to evaluate whether there are pending jobs whose location necessitates triggering some instance of *GrabJob*. This gives us an example of a percept and a query required in MATSim. On the right hand component in Figure 31.2, we see four different actions which will have a corresponding BDI-action on the MATSim side. We will focus here on the *DriveTo* action, but the *PickUp* and *DropOff* would be realized in a similar way, using MATSim activities rather than legs.

The following must usually be done:

- Every plan trigger which is information from MATSim must be defined as a percept.
- All information required from MATSim, that is not a trigger, must be defined either as a percept (and then stored locally), or as a query.
- All actions which should be executed in MATSim must be defined.

In the rest of this section, we describe exactly what must be provided in the MATSim application files for each of these to work as expected. Instructions and examples for the BDI application can be found in the integration repository (noted at start of chapter).

31.4.1 The *ClosetoDest* Percept

All functions for collecting percepts for the BDI system are defined in the *AgentActivityEventHandler* class. Perusal of existing functions can ascertain whether the desired percept is already calculated. For example, *arriveAtDest* is already defined for use as a BDI percept. If the percept collection function already exists, the developer must ensure that the appropriate agent type is registered for this percept within the relevant function. For example, in *arriveAtDest()* we have:

```
if agent.type = taxi
  AND agent.loc = dest(agent) \* obtained from infrastructure data *\
  // collect and package this percept
```

If we now want this percept provided to agents of type *commuter*, we must make the first line:

```
if ((agent.type = taxi) OR (agent.type = commuter))
  AND agent.loc = dest(agent) \* obtained from infrastructure data *\
  // collect and package this percept
```

The `arriveAtDest` function is triggered by the MATSim `LinkEnterEvent` event using MATSim provided callbacks. Thus, we have defined `handleEvent(LinkEnterEvent)` to call all percept collection functions triggered by this event – in this case `arriveAtDest`.

The `ClosetoDest` percept will be triggered by the same MATSim event `LinkEnterEvent`, so to add this, we must add the call to `ClosetoDest` in the `handleEvent(LinkEnterEvent)` and then define our `ClosetoDest` function within the `AgentActivityEventHandler` class. We only want to send the `ClosetoDest` percept when we first come within the defined distance of our destination, not at every step. Therefore, the `ClosetoDest` function must first check whether this percept has already been sent to this agent, for the current destination. If so, nothing more is done. If not, it is ascertained whether the link entered is within the desired “close-to” distance and, if so, the percept is registered. For efficiency, the first link “close-to” the dest can be calculated and recorded when the `DriveTo` action is initiated; in which case, one must only check whether the entered link-ID is the same as the recorded “close-to” link-ID.

In principle, percepts could also be calculated in a function executed after all agents had been stepped. The important thing is that when a percept occurs, it is recorded in the percept data package for that agent. Further work is required to ascertain which percept collection methods will be most efficient with very large numbers of agents.

31.4.2 *The RequestLocation Query*

Queries are defined in, and managed through, the `MATSimPerceptQueryHandler` class. A function `queryPercept(agent, query, response)` responds to a query by extracting the specific query and calling the relevant defined function. So, for example, to respond to the `queryPercept(ownID, ‘RequestLocation agentID’, loc)` query from an agent, `queryPercept` will contain the code:

```
if percept_type = "RequestLocation"
    loc = getLocation("agentID")
```

The `getLocation` function will then ascertain the location of `agentID`, storing the value in `loc`. If the query is already defined in MATSim, nothing further is required to use it in an application.

31.4.3 *The DriveTo BDI-Action*

The `DriveTo(loc)` BDI action is, of course, the most basic and commonly used BDI action in MATSim and is already implemented in our infrastructure. As long as the appropriate BDI action and parameters are passed in the information package from the BDI system, nothing further is required within MATSim. However, for the purpose of illustration, we will assume it has not yet been implemented and we will go through the steps of defining a new BDI action with this as an example.

The `MATSimActionList` class defines mappings for all BDI actions in the system and the MATSim function calls that realize those BDI actions. Any new BDI action must first be added to this list.

The `MATSimActionHandler` defines all functions that realize BDI actions, as well as a `processAction` function which handles all BDI action strings from the BDI system, calling the appropriate helper functions. Thus any new BDI action must have its implementation defined within this class and must have the appropriate call to the function added within `processAction`. Let us call the relevant function that we will add `processDriveTo`. This function will always need the `agentID` as a parameter, as well as whatever parameters are provided in the action package.

So, in our example, we will have the function `processDriveTo(agentID, loc)` which needs to be defined. The function for the new action must perform two key tasks:

1. Obtain the MATSim plan of the relevant agent and modify it so that regular MATSim execution of the plan will have the desired effect.
Generally, when the plan is accessed, it will have a single dummy activity with end-time infinity. The end time of this activity must be set to now and a leg must be instantiated with the link corresponding to the destination `loc` as the end point and the links to be followed, as calculated by the router. This leg must then be inserted into the plan, followed by a new dummy activity instance with end time infinity.
2. Place the action instance into the list of actions being monitored.

It is also necessary to set up recognition of when the action has finished, so that this information can be sent back to the BDI system and the agent can continue to reason about its next actions. This is done via the MATSim callbacks provided, in the same way as detecting percepts. However, the corresponding function, instead of placing information in the percept package for the agent, will modify the status of the relevant BDI action instance in the information package to `PASSED` and remove the instance from the list of actions being monitored. It is also possible to define a condition where the action should be considered `FAILED` and to detect this in a similar way. Alternatively, failure can be managed by sending a percept, and having the BDI agent abort the action as a result⁵.

The current structure assumes that multiple actions of a single agent cannot be executed in parallel (a reasonable assumption for MATSim). It is the responsibility of the BDI system to allow only one active BDI action per agent.

Further instructions, as well as examples, can be found in our BDI-MATSim integration repository.

31.4.4 Discussion

An important aspect of a simulation design using BDI agents within MATSim is deciding on which abstraction level BDI actions should be described. So far, we have tended to have BDI actions map to a single leg or activity within a MATSim plan. However, it is certainly easy to think of BDI actions that combine several such components. Straightforward examples would be grocery shopping or taking kids to school - both involving a leg to a destination, an activity at that destination and a return leg. There are no immediately obvious advantages associated with BDI actions at higher abstraction levels (requiring coding of these actions in MATSim) vs using lower level BDI actions with the higher level coded as BDI plans/goals. Future experience and experimentation may provide insights to guide decisions.

31.5 Examples

Here, we describe two different examples of BDI agents within MATSim: a bushfire evacuation simulation, where MATSim is being used because traffic flow is a crucial component in this type of evacuation and a taxi application developed as a demonstrator for integration of a BDI system with MATSim (Padgham et al., 2014). We compare this approach to incorporating taxis with that described in Chapter 23 for incorporating dynamically scheduled vehicles and with the approaches to “within-day replanning” described in Chapter 30.

⁵ The simplest way in JACK is to use a maintenance condition relying on a belief that is modified as the result of a percept.

Both our example applications use only the Mobsim engine (QSim) of MATSim and do no repeated daily cycles with plan scoring and modification. There are undoubtedly applications which could benefit from a combination of BDI agents and agents which evolve using MATSim's scoring and replanning, but we have not yet investigated them.

31.5.1 *Bushfire Example*

The bushfire example (currently) involves modeling of residents and their decision-making behavior about what to do regarding a nearby bushfire. Potential driving activities include picking up children from a school or other facility, checking on neighbors or friends and driving to a local or more distant destination, possibly via a specified route. Decision making may involve various factors, such as time of day, ideas about what other family members are doing, warnings and notifications from emergency services, observations of neighbors, etc. In one approach, we focused on incorporating well-developed and validated actual human decision making models in a bushfire situation, developed by a collaborator. Our contribution has been to integrate this with MATSim, using our integration framework, to provide data about any traffic-related issues, thus providing a more valuable simulation to planners. In our other approach, we model both residents and an incident controller. Here, our focus has been on technical issues that involve providing an interactive simulation suitable for use by emergency services personnel and/or communities for exploration of potential strategies.

In the interactive version, the incident controller assigns specified evacuation centers and routes to residents in certain sections of the town being evacuated. Evacuation of different areas may be started at different times. Residents follow the incident controller's instructions with some probability based on their individual situations (currently modeled very superficially). Following the suggested route is achieved by driving via suggested way points (using the *DriveTo* BDI action), with the BDI agent (potentially) re-assessing as each waypoint is reached. An alternative would be to define a new BDI action *DriveToViaWaypoints*. One issue that arose during the development of this simulation involved road congestion; MATSim routing algorithms began developing very circuitous routes, sometimes going back towards the fire threat. There were two issues illustrated here about developing a realistic simulation: one was that, realistically, people would not choose their routes based on global knowledge of current congestion; the other was that, regardless of congestion, people would not head back into the fire zone. The current solution is to use a routing algorithm not accounting for current road speeds, using only static speed limits. Going forward, one may want to assume some knowledge of congestion (based on radio broadcasts or other social media). An interesting future research question is how to best achieve responsibility sharing for realistic behavior between MATSim and the BDI decision-making program, on route selection.

31.5.2 *Taxi Example*

The taxi prototype application was developed purely as a 'proof of concept', allowing decisions to be made dynamically by the BDI brain on an ongoing basis, then carried out by the MATSim execution engine. There is a simple taxi administrator in the BDI system, which generates jobs, posts them to a notice board and confirms requests from taxis to take specific jobs. Taxis have plans allowing them to take jobs from the board, go to a taxi rank, or take a break. After taking a job from the board, the taxi drives to the pick-up address, picks up the passenger, then drives to the destination and drops them off. When the taxi approaches the destination, it looks on the job board for nearby jobs; if something suitable is found, it requests it from the administrator. The only BDI action implemented in this application is a simple *DriveTo*. The *ClosetoDest* percept was used as described in Section 31.4. This application was tested with the Berlin road network and the 15 963 agents in the MATSim sample files, with all agents operating as BDI taxi agents. Profiling

showed that, by far, the majority of the execution time was spent in route planning, with very little in the BDI reasoning, or communication with the BDI system.

31.5.3 Discussion

Both evacuation and taxis are discussed in Chapters 30 and 23, as applications requiring a reactive approach to planning, rather than iteration over many days to find the preferred plan. Chapter 23 discusses two implementation options: one which replaces the MATSim agent with an agent that considers what to do at each relevant decision point (particularly intersections); the other leaves the agent code as is, but modifies the agent's plans when certain events occur. The BDI approach has the computational advantages of the latter, in that only a small subset of agents require changes to their plans at any simulation step and many existing MATSim routines can be used to modify the plans. However, it also has many of the advantages of the former approach; agents are still fully autonomous, with all decision making occurring within the BDI system. By registering for any percepts which could potentially cause the agent to change its mind, the agent remains fully in control at all times. However, it only needs to decide its next action when it completes the current high level action—which will almost certainly be orders of magnitude less often than at each intersection—or when a percept arrives indicating a need to reconsider. The provision of the ability to drop current BDI actions (legs or activities) provides the same level of reactive autonomy as the fully reactive within day replanning agent, but probably at a lower computational cost. Perhaps more important than the computational cost savings: agent decision making can be programmed in a framework that is at a high level of abstraction, using goals, plans and beliefs, within existing highly efficient platforms such as JACK (Winikoff, 2005), Jadex (Braubach et al., 2005) or Jason (Bordini et al., 2007). Design tools for developing such agents also already exist (Padgham and Winikoff, 2004). One study has shown that using a BDI language makes program development hugely more efficient than programming in Java (Benfield et al., 2006). The close mapping between intuitively understandable design diagrams and the program code implementing this in a BDI system is also highly advantageous for validating design of realistic agents with domain experts. We have discussed design of resident agents in a sandbagging flood scenario, with emergency services personnel extremely experienced in that domain and found the representation to be effective. We consider that this representational aspect can be a significant advantage when compared to programming the agent using the `DynAgentLogic` facility described in Chapter 23.

SUBPART EIGHT

Automatic Calibration

CHAPTER 32

CaDyTS: Calibration of Dynamic Traffic Simulations

Kai Nagel, Michael Zilske and Gunnar Flötteröd

32.1 Basic Information

Entry point to documentation:

<http://matsim.org/extensions> → `cadytsIntegration`

Invoking the module:

<http://matsim.org/javadoc> → `cadytsIntegration` → `RunCadyts4CarExample` class

Selected publications:

Flötteröd (2010); Flötteröd et al. (2011); Flötteröd et al. (2011a); Flötteröd (2008); Moyo Oliveros (2013)

32.2 Introduction

Cadyts (Calibration of Dynamic Traffic Simulations)¹—licensed under GPLv3 (GNU General Public License version 3.0)—calibrates disaggregate travel demand models of DTA (Dynamic Traffic Assignment) simulators from traffic counts and vehicle re-identification data. Cadyts is broadly compatible with DTA microsimulators, into which it can be hooked through parsimonious interfaces.

As explained formally in Chapter 47 and 48, DTA aims at consistency between a dynamic travel demand model, defining the choice of activity-travel plans, and a dynamic network supply model, capturing spatiotemporal network flows and congestion evolution.

¹ <http://people.kth.se/~gunnarfl/cadyts.html>

How to cite this book chapter:

Nagel, K, Zilske, M and Flötteröd, G. 2016. CaDyTS: Calibration of Dynamic Traffic Simulations. In: Horni, A, Nagel, K and Axhausen, K W. (eds.) *The Multi-Agent Transport Simulation MATSim*, Pp. 213–216. London: Ubiquity Press. DOI: <http://dx.doi.org/10.5334/baw.32>. License: CC-BY 4.0

Cadyts adjusts the plan choice probabilities of all agents, resulting in simulated network conditions that are consistent with measured real-world data while maintaining the behavioral plausibility of the underlying travel demand model. Within MATSim, plan choice probabilities adjustment is realized by adjusting plan scores, as explained in the next section.

32.3 Adjusting Plans Utility

When traffic counts are the empirical source, plan-specific score corrections are composed of link- and time-additive terms $\Delta S_a(k)$ for each link a and each calibration time step k (often one hour). When congestion is light and traffic counts are independently and normally distributed, these correction terms become

$$\Delta S_a(k) = \frac{y_a(k) - q_a(k)}{\sigma_a^2(k)} \quad (32.1)$$

where $y_a(k)$ is the real-world measurement on link a in time step k , $q_a(k)$ is its simulated counterpart and $\sigma_a^2(k)$ is (an estimate of) the real measurement variance (assuming its expected value coincides with the prediction $q_a(k)$ of a perfectly calibrated simulator).

The score correction of an agent's given activity-travel plan is calculated as the sum of all $\Delta S_a(k)$, given that following that plan implies entering link a within time step k . With this, the *a posteriori* choice probability of agent n 's plan i given the count data $\mathbf{y} = \{y_a(k)\}$ becomes

$$P_n(i | \mathbf{y}) \sim \exp \left(S_n(i) + \sum_{ak \in i} \Delta S_a(k) \right) = \exp \left(S_n(i) + \sum_{ak \in i} \frac{y_a(k) - q_a(k)}{\sigma_a^2(k)} \right) \quad (32.2)$$

where $S_n(i)$ is the *a priori* score of plan i of agent n , as calculated for example with Equation (3.1) and $ak \in i$ reads: "following plan i implies entering link a in time step k ".

Intuitively, if the simulated value $q_a(k)$ is smaller than the real measurement $y_a(k)$, then a score increase, and thus a choice probability increase, results. The variance $\sigma_a^2(k)$ denotes the level of trust in that specific measurement—a large $\sigma_a^2(k)$ implies a low trust level, taking effect through a large denominator in the corresponding score correction addend.

Flötteröd et al. (2011) is the key methodological reference on Cadyts. It derives the calibration approach from a Bayesian argument and provides more technical information, such as a more general correction of the utility function than in Equation (32.1) that also applies when congestion is present. A lighter presentation is Flötteröd et al. (2011a), where the formulas above are discussed in somewhat greater detail.

32.4 Hooking Cadyts into MATSim

Hooking Cadyts into MATSim is based on the following operations:

1. Initialization: When the calibration is started, it requires all available traffic counts and some further parameters. For this, the Cadyts function `void addMeasurement(...)` is called once for every measurement before the simulation starts. It registers a certain measurement type, which has been observed on a specific link.
2. Iterations: The calibration is run jointly with the simulation until (calibrated) stationary conditions are reached.
 - a. Demand simulation: The calibration needs an access point in the simulation to affect the plan choice. There are various ways to realize this, depending on the simulator. Before a MATSim agent chooses a plan, it asks the calibration through the Cadyts function

```
double calcLinearPlanEffect(cadyts.demand.Plan<L> plan)
```

for all of this plans' score offsets. The agent then chooses a plan based on accordingly modified scores.

All selected plans of an iteration are registered to Cadyts by

```
void addToDemand(cadyts.demand.Plan<L> plan) .
```

Since Cadyts has its own plans format, MATSim plans need to be converted to that format beforehand.

- b. Supply simulation: The calibration must observe simulated network conditions to evaluate their deviation from real traffic counts. For this, the Cadyts function

```
void afterNetworkLoading(SimResults<L> simResults)
```

is called once after each network loading. It passes a container object to the calibration that provides information about the most recent network loading results, particularly on simulated flows at measurement locations.

32.5 Applications

Cadyts has been successfully applied in studies like Ziemke et al. (2015); Zilske and Nagel (2015); Flötteröd et al. (2011a). Zürich scenario results illustrate its efficiency, as shown in Flötteröd et al. (2011b, Slide 8), reproduced in Figure 32.1.

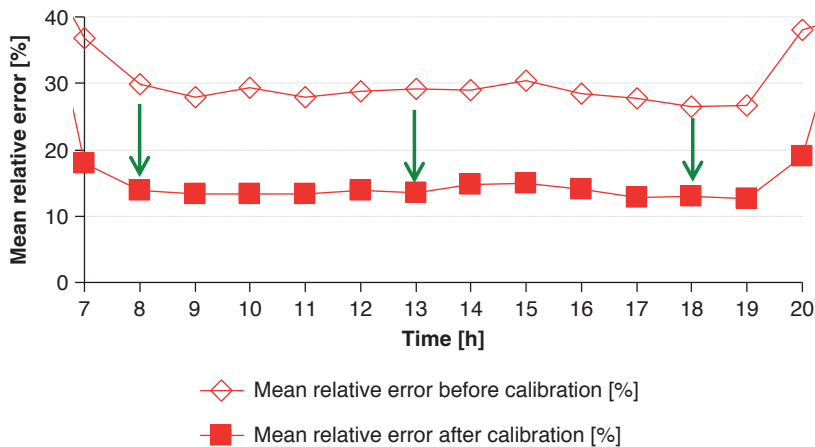


Figure 32.1: Zürich case study results: mean relative error in link volumes.

Source: Flötteröd et al. (2011b, Slide 8)

SUBPART NINE

Visualizers

CHAPTER 33

Senozon Via

Marcel Rieser

33.1 Basic Information

Entry point to documentation:

<http://matsim.org/extensions> → Via

Invoking the module:

Standalone GUI, double-clickable jar file

Selected publications:

<http://via.senozon.com> → Download → manual

33.2 Introduction

Via is an application to visualize and analyze MATSim simulation results. Unlike MATSim, *Via* is not open source; it is developed as a proprietary commercial software by Senozon AG, an ETH Spin-off company founded by two former PhD students involved in MATSim development. Shortly after the company was founded, first (potential) client presentations began; the lack of visual material was an obvious handicap. Explaining to customers that all answers to their questions were contained in a huge events file was not satisfactory; pictures or even animations made it much easier for them to understand. Thus, work on a visualization tool started as soon as the company was set up. Initially planned as a purely internal tool, it quickly became clear that a graphical visualization and analysis tool would also benefit other users of MATSim. After a beta test phase with selected MATSim users in Spring 2011, the first version of *Via* was released in July 2011. Since then, the list of features provided by the application has grown continuously.

Via is written in Java and thus works on any platform able to run MATSim. For easier deployment, the application comes as double-clickable, native executable on Windows and Mac OS X,

How to cite this book chapter:

Rieser, M. 2016. Senozon Via. In: Horni, A, Nagel, K and Axhausen, K W. (eds.) *The Multi-Agent Transport Simulation MATSim*, Pp. 219–224. London: Ubiquity Press. DOI: <http://dx.doi.org/10.5334/baw.33>. License: CC-BY 4.0

partially hiding its Java nature. A limited version is available for free and can be downloaded from the product website (senozon AG, 2015). Different licenses are available for commercial usage or for research or educational purposes to serve different user group needs.

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

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

33.3 Simple Usage

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

By default, *Via*'s window looks similar to the one shown in Figure 33.1. To add a file as a data source, the file can either be drag-and-dropped onto the layers list left of the black visualization area, or by choosing Add Data... from the File menu. To add a layer, the little plus icon in the lower left of the window can be pressed, or by choosing Add Layer... from the File menu. To get started, it's usually best to add a network and (small) events file from MATSim to *Via*, and create a Network layer and a Vehicles layer.

Elements shown in the visualization area like the network or vehicles can be queried. Queries are usually provided by layers, made available with buttons with question-mark icons. Clicking

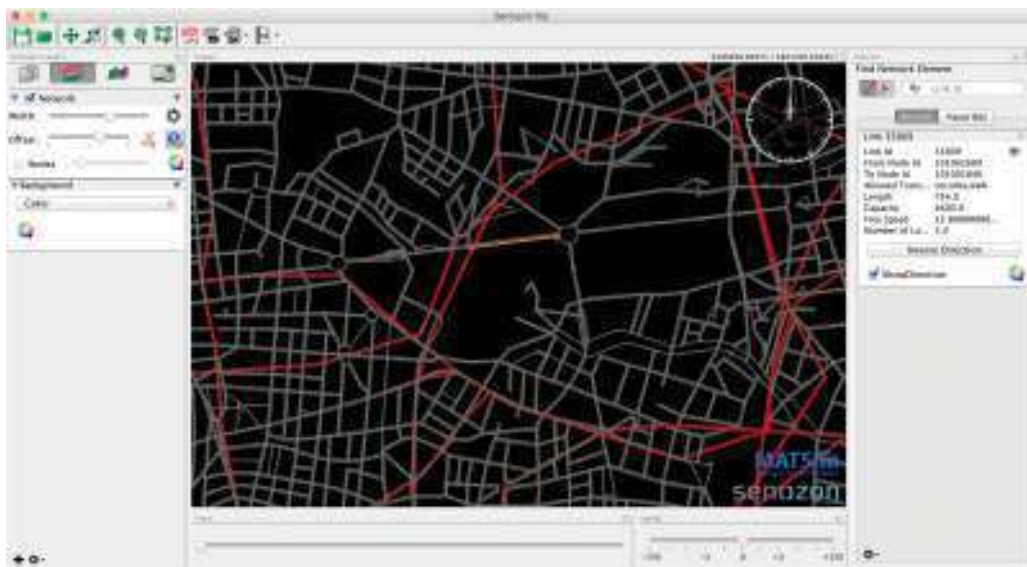


Figure 33.1: Via's window with default layout and a network query being shown.

such an icon activates the corresponding query mode, and any subsequent click on the visualization area will run the query. Query results are shown on the right side of the visualization area. Figure 33.1 shows a network query for links. One query is special, globally available, and not linked to a layer: querying an agent plan. This query is available from the toolbar, next to the icon, to shift the visualization view around.

Once a query has been made, *Via* often allows another query based on the current query results. By right-clicking in the visualization area, a pop up menu appears with more options regarding the last query, as well as additional possible queries. Examples are: Select Link Analysis given a link, Select Facility Analysis given a facility, List Transit Lines that use a given link, or List Passengers if a transit vehicles was queried in the first place.

33.4 Use Cases and Examples

33.4.1 Agent Visualization

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

The visualization can be further customized; with the addition of a `population.xml` file, more detailed activity coordinates can be loaded to obtain a better distribution of activity locations (MATSim's events file does not contain coordinates for activities, only the assigned link ID. So by default, all activities taking place on a link are first shown at the location of the link's to-node). Vehicles and groups of vehicles can also be styled differently; it is possible to visualize transit vehicles with a square shape with colors representing the occupancy of the vehicles, pedestrians or cyclists in a multi-modal simulation can be shown as circles and private cars can be displayed with a triangular shape with colors representing their absolute speed or their speed relative to the allowed maximum speed on their current link (see Figure 33.2). As mentioned above, arbitrary groups of vehicles can be styled differently, which is useful to highlight special agents, e.g., when simulating a fleet of electric vehicles, a car sharing fleet, or agents simulated with special routing guidance.

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

33.4.2 Facility Analysis

Activity facilities allow for very detailed modeling in MATSim, especially considering the functionality provided by the destination innovation module (Chapter 27). *Via* provides several unique ways to analyze the mobility effects to and from facilities.

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

An alternative analysis is similar to the—for transport planners—well known Select Link Analysis, but designed for facilities: the Select Facility Analysis. This analysis shows the combined link loads produced by agents arriving or departing at a facility, showing the starting location for agents visiting a specific facility and what routes they use. Figure 33.3 shows such an example.

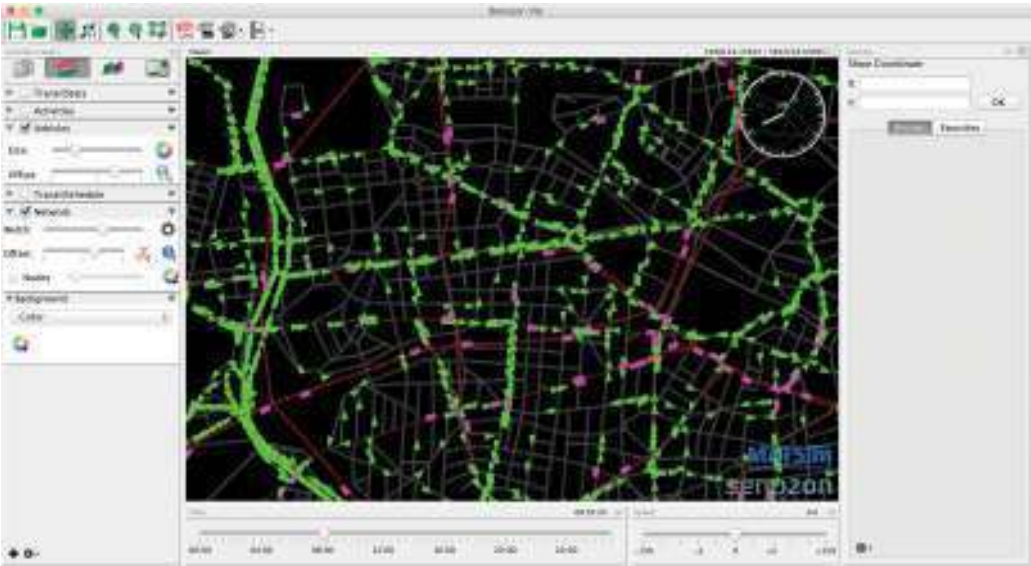


Figure 33.2: Vehicles in Via: Green triangular symbols represent private cars, pink rectangular symbols public transport vehicles.

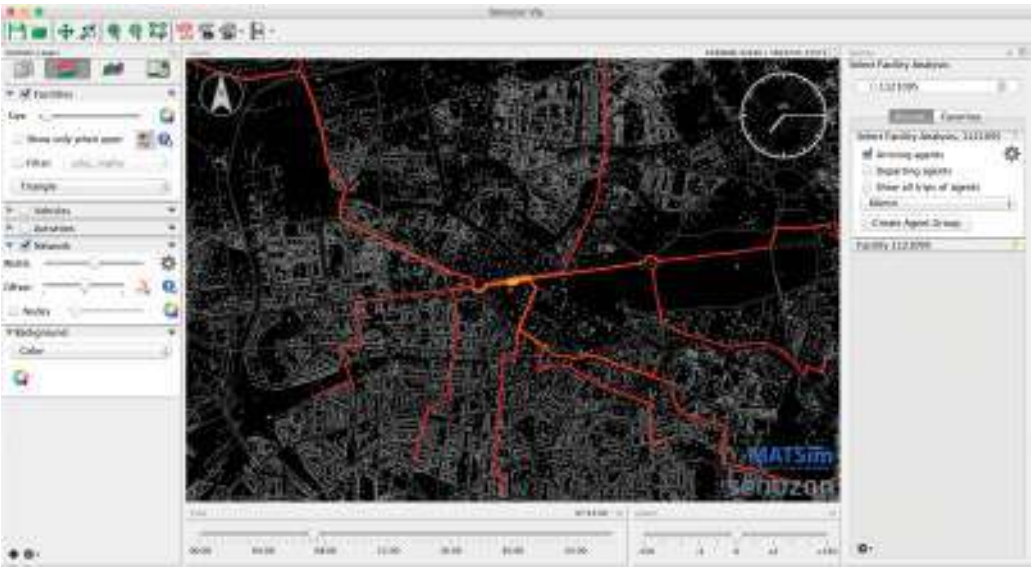


Figure 33.3: Select facility analysis: Links used to travel to and from a facility are highlighted.

33.4.3 Public Transport Analysis

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



Figure 33.4: Passenger flows on a transit line.

A special, but very useful visualization is the Route Flow analysis. This shows, in a visually appealing way, the number of passengers traveling between two stops along a route—for all possible stop combinations. Figure 33.4 shows an example of such a route flow with the route of the transit line shown in the background. It is clear that the demand on the bus route is more or less split in two; a first travel demand up to about the first third of the route, and then it again collects passengers all wishing to go to one of the last stops along the route. This could indicate that it might make sense to split the line in two.

33.4.4 Scenario Comparisons

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

33.4.5 Aggregating Data

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

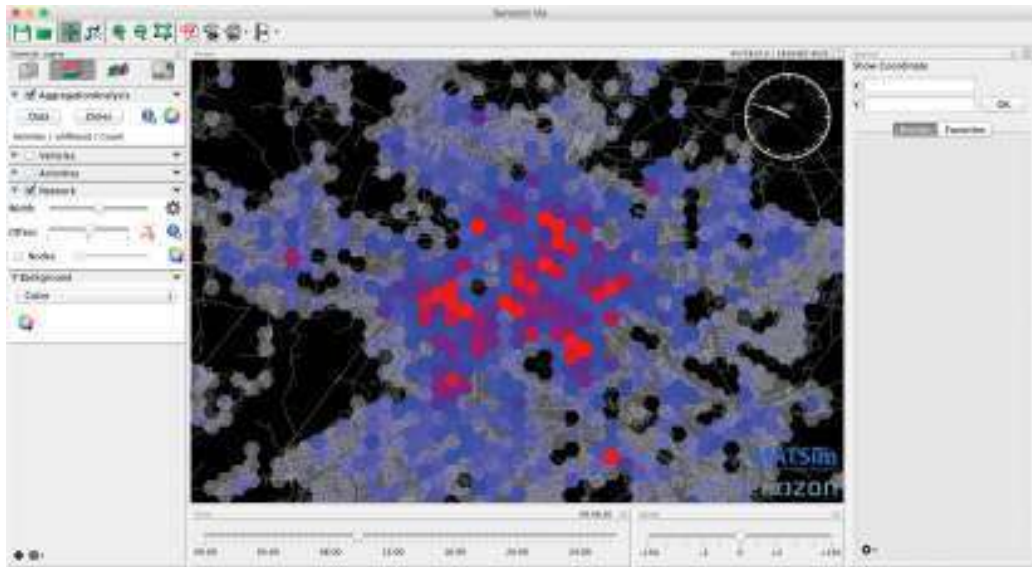


Figure 33.5: Aggregation analysis: Number of performed activities during the whole day.

for aggregation, and the trips performed by agents as O-D data sources, any tabular custom data with coordinate attributes can also be used for this.

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

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

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

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

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

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

CHAPTER 34

OTFVis: MATSim's Open-Source Visualizer

David Strippgen

34.1 Basic Information

Entry point to documentation:

<http://matsim.org/extensions> → otfvis

Invoking the module:

<http://matsim.org/javadoc> → otfvis → OTFVis class, RunOTFVis class

Selected publications:

Strippgen (2009)

34.2 Introduction

For most MATSim users, Via's (Chapter 33) free branch will be a good solution for their visualization needs. However, if project demand reaches beyond the given (and fixed) abilities of the Via free version, there is another—though not as stylish—option for MATSim output visualization, the OTFVis.

The short term for “On the Fly Visualizer”, OTFVis was designed to support actual visualization of live simulation runs with MATSim. Therefore, one purpose of the OTFVis is the debugging of MATSim (input) data. Nonetheless, playing prerecorded movie (MVI (An OTFVis Movie File, not to be confused with the “Musical Video Interactive” file usually abbreviated mvi)) files created from MATSim events is another way to use OTFVis. Generally speaking, OTFVis serves as an open-source counterpart to the possibilities Via gives the MATSim community. The OTFVis is written in Java and available as source code to extend for different MATSim projects' special needs. Hence, it is possible and desirable to actually extend the OTFVis functionality, incorporating the user's own data sets and visualizations.

How to cite this book chapter:

Strippgen, D. 2016. OTFVis: MATSim's Open-Source Visualizer. In: Horni, A, Nagel, K and Axhausen, K W. (eds.) *The Multi-Agent Transport Simulation MATSim*, Pp. 225–234. London: Ubiquity Press. DOI: <http://dx.doi.org/10.5334/baw.34>. License: CC-BY 4.0

34.3 Using OTFVis

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

34.3.1 MVI Files

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

34.3.2 Starting OTFVis

OTFVis is a MATSim contribution. There is no actual stable release of the OTFVis package; so, to acquire a working version, a “nightly build” needs to be downloaded as shown in Section 44.3.6. There, one finds the latest `otfvis-version-SNAPSHOT-build.zip` file available for download. Unzip it to the place where the `matsim.jar` already resides; do not forget to extract the `libs-directories` found in the respective zip files.

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

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

(exchange “;” with “.” depending on the used OS (Operating System)) is a good starting point. This will open the dialog window shown in Figure 34.1, asking for one choice from four possible usages of OTFVis; these will be explained in the next section.

34.3.3 Use Cases of OTFVis

With the open dialog appearing after starting the vanilla OTFVis class, the following options appear, as shown in Figure 34.1:

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

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

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



Figure 34.1: OTFVis Start Dialog.

34.3.3.1 *Converting Event Files*

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

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

34.3.3.2 *Network File Loading*

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

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

34.3.3.3 *Running a MATSim Configuration*

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

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

34.3.3.4 Loading & Displaying an MVI File

If the first and default option tab is chosen, a MVI file is selected and shown as detailed in next section 34.3.4. This is the most common use case for OTFVis; the same results can be achieved by starting OTFVis from the shell with an MVI file as an argument.

34.3.4 Viewing an MVI File

An example is illustrated in Figure 34.2. On the top left of the application, one finds buttons for controlling the file playback. A short summary of the functionality is given in Table 34.1.

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

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

34.3.5 General Interaction with the Main Screen

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

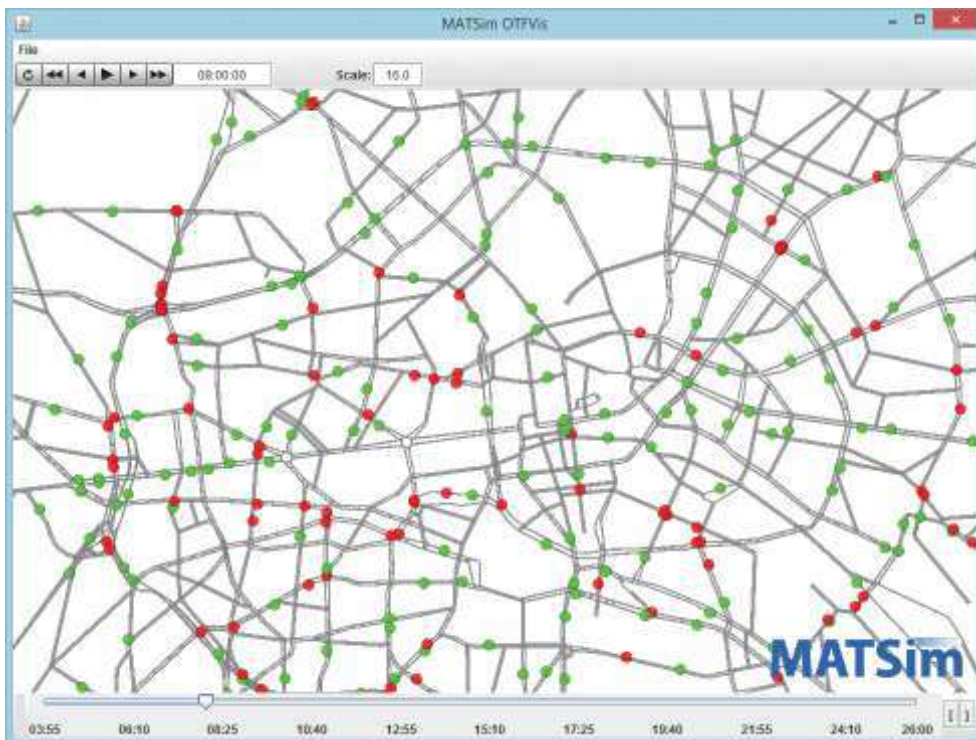


Figure 34.2: Displaying an MVI file.








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

Table 34.1: OTFVis Buttonbar.

- Right button drag:** Extend a rectangle for zooming into the view. Releasing the button will execute a zoom, so the chosen rectangle will best fit the screen.
- Middle-Mouse-drag:** Pan (translate) the screen.
- Right-Mouse-Click:** Show a context menu (for now only with the option to save the view settings).

34.3.6 User Interaction in the Live Mobsim

When started as a live simulation, OTFVis will look different than Figure 34.3. First, the controls of the simulation's view flow are a restricted subset of those used in MVI playback. There is no way to reset or rewind the simulation. One can still take small or large steps forward. A new option

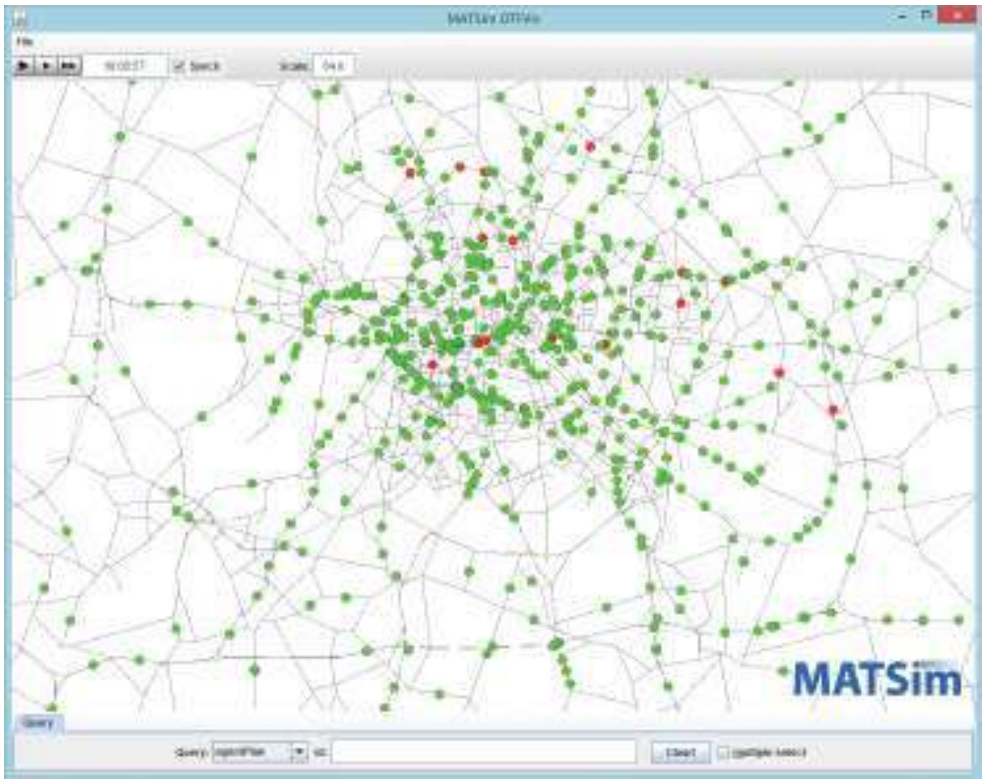


Figure 34.3: Live mode.

is given by the synch checkbox, which determines whether the mobsim will stop for each frame the OTFVis renders, or run independently. Usually the un-synched version will proceed faster, as the OTFVis output is restricted to a default of about 30 frames/updates per second and a small mobsim's simulation speed will be a magnitude higher. The time-consuming generation of visualization data will also only be necessary for a small fraction of the simulation. Length of OTFVis pauses between frames can be configured in the preferences dialog.

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

34.3.7 Running a Query in OTFVis Real Time Data

From the dropdown box, one can choose the different query types. Often, additional input is necessary, either in the text field next to it or, more often, by clicking into the network. To give an example with agent query selected, a click onto any agent's symbol will give a visualization of this particular agent's day plan. This is shown in Figure 34.4. There are other pre-defined queries. These queries are rather project-oriented, so defining own queries will probably be necessary to make best use of this option. In the second part of this chapter, we will look into defining own queries.



Figure 34.4: Queries.

34.4 Extending OTFVis

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

34.4.1 Design Principles of OTFVis

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

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

MATSim runs can easily engage millions of agents traveling a network. To make a visualization of these large data sets feasible, two measures have been taken. A quad tree structure was implemented to ensure that only the smallest set of data necessary to display the visible sector of the network is transferred. The quad tree is a simple data structure to aggregate spatial data and retrieve parts of it efficiently for real time visualization. Apart from data structures, hardware is also used to speed up displaying the simulation. OpenGL is a platform-independent API for interfacing graphics hardware, specifically the 3D acceleration chips implemented in every contemporary computer. With the aid of 3D graphics hardware, millions of agents can be displayed in real time. Other measures were taken to segregate data extraction from data visualization, like the reader/writer pairs presented in the next section.

34.4.2 Readers and Writers

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

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

For each writer, there has to be a sibling-reader class, responsible for reading back extracted data from the byte buffer. It is crucial to ensure that these pairs work synchronously. Most

Writer/Reader-pairs are implemented in the same class, since having the source-code at the same place reduces errors in the synchronization.

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

34.4.3 *Visualization of the Data*

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

34.4.4 *Layers*

To make sure that only data actually necessary for drawing the particular area visible in the viewport is sent, writers should minimize the data packets, so the quad tree can make a spatial data reduction. This seems somewhat in opposition to OpenGL or any graphics API. The API wants maximal data to be accumulated, to optimize output through the underlying hardware graphics pipeline. Think of an assembly line vs. a handcrafted item; whenever the flow of data is interrupted, the assembly line stalls and graphics performance derogates. To ease this issue, “layers” have been introduced to OTFVis. Any drawer (responsible for a bit of information) can be assigned to a layer and these layers will ultimately be summoned to draw the screen’s content. It is up to the layer to optimize the execution of the drawers when necessary. For example, a network layer might store all network info from the drawer in one array, or display a list to optimize drawing of the network; (often in OpenGL, it is advisable to rather let the hardware decide what to draw. It might be faster to have all complete data residing in graphics hardware memory, rather than to transfer the reduced information set every frame). There are three layers predefined in OTFVis. The *networkLayer* contains the static street net, the *agentLayer* the actual dynamic agents and a third layer, the *miscellaneousLayer*, contains additional data.

34.4.5 *Patching the Connections*

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

This class maps several routes for the information coming from the mobsim, building a chain of responsibility. Each data item starts at a link in our network. An *OTFDataWriter* object is responsible for extracting the desired data from the link and writing it into a *ByteBuffer*. Complementing this, an associated *OTFDataReader* is needed to retrieve data from the buffer. This item will also be responsible for adding a drawable item derived from the

class `OTFGLAbstractDrawable` to the scene graph representing the actual screen content. The connection between these items is made by adding entries into the `OTFConnectionManager`, with calls to `OTFConnectionManager.connectLinkToWriter(OTFDataWriter)` and `OTFConnectionManager.connectLinkToWriter(OTFDataWriter, OTFDataReader)`, respectively.

Example (from the `OTFClientLive.java`):

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

    OTFLinkAgentsHandler.class);
```

34.4.6 *Sending the Data*

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

34.4.7 *Performance Considerations*

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

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

34.4.8 *Sending Live Data*

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

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

SUBPART TEN

Analysis

CHAPTER 35

Accessibility

Dominik Ziemke

35.1 Basic Information

Entry point to documentation:

<http://matsim.org/extensions> → accessibility

Invoking the module:

<http://matsim.org/javadoc> → accessibility → RunAccessibilityExample class

Selected publications:

Nicolai and Nagel (2014); Joubert et al. (2015)

In transport science and planning, the term accessibility can refer to at least three different concepts. First, accessibility may be used to describe how well a certain transport infrastructure component can be utilized by travelers, particularly those with handicaps (Faura, 2012). In this sense, *accessibility guidelines* tell engineers and planners how to design transport infrastructure elements, such as public transport facilities, to make them accessible, i.e., useable for all travelers. Second, accessibility may be used to describe how easy/convenient the approach to a given land-use facility is. There are, for instance, studies (Fujiyama, 2004) to improve the accessibility of shopping centers by redesigning access roads and their connection to major roads. Finally, the term accessibility can be used in a more global way, to describe availability and spatial distribution of activity facilities within a given area, e.g., a metropolitan region and the ease with which these facilities can be reached from other locations in the area. MATSim's accessibility extension focuses on all these aspects; the discussion in this chapter draws on Nicolai and Nagel (2014).

How to cite this book chapter:

Ziemke, D. 2016. Accessibility. In: Horni, A, Nagel, K and Axhausen, K W. (eds.) *The Multi-Agent Transport Simulation MATSim*, Pp. 237–246. London: Ubiquity Press. DOI: <http://dx.doi.org/10.5334/baw.35>. License: CC-BY 4.0

35.2 Introduction

Improvement in accessibility is often defined as a central goal of proposed transport or infrastructure schemes (Geurs et al., 2012b) and accessibility is usually a precisely-defined, quantitative measure. While Batty (2009) traces the origins of the accessibility concept back to location theory and regional economic planning in the 1920s (when transport planning began in North America; Geurs et al., 2012b), Hansen, with his widely-cited paper (Hansen, 1959), is generally credited with the first real definition of accessibility, defining it as *the potential of opportunities for interaction*. In more detail, Morris et al. (1979) define accessibility as “the ease with which activities may be reached from a given location using a particular transportation system”. The concept of accessibility is a potential methodology for the assessment of transport systems, as it is a comprehensive and inclusive way to evaluate how, where and why people move, taking well-known dependencies between transport and land use into account. Hansen (1959) was probably the first to develop a procedure for quantitative consideration of accessibility, discussed in more detail in Section 35.3.

In their widely-cited review, Geurs and van Wee (2004) identify four accessibility components from existing definitions and applied measures:

1. The **land-use** component reflects the number and spatial distribution of opportunities.
2. The **transport** component describes the effort to travel from a given origin to a given destination.
3. The **temporal** component considers the availability of activities at different times of day, e.g., during morning peak hours.
4. The **individual** component addresses various socio-economic groups’ different needs and opportunities, e.g., different income groups.

In this review, Geurs and van Wee (2004) list and summarize typical approaches applying the accessibility concept, focusing on the accessibility components discussed above:

1. **Infrastructure-based** measures focus on the (observed or simulated) performance or service level of transport infrastructure, e.g., represented as average travel speed. These measures are typically used in transport planning.
2. **Location-based** measures describe level of accessibility to spatially distributed activities, such as number of jobs within 30 minutes travel time from origin locations. These measures are typically used in urban planning and geographical studies.
3. **Person-based** measures analyze accessibility at the individual level, such as the activities in which an individual can participate at a given time. These measures are grounded in Hägerstrand (1970)’s space-time geography.
4. **Utility-based** measures analyze the economic benefits that people derive from access to spatially distributed activities. These measures have their origin in economic studies.

Geurs and van Wee (2004) intersects these approaches with the four accessibility components identified above, creating a matrix. This matrix illustrates how each of the four accessibility components is represented in the four different accessibility measures. There, each measure focuses on certain weaknesses in those accessibility components outside the focus of a specific measure. Accordingly, Geurs and van Wee (2004) recommend that an accessibility measure include all four discussed accessibility components. The accessibility extension of MATSim, described in the following, could be one way to achieve this goal.

In other recent research, as identified by Geurs et al. (2012b), the accessibility concept is also applied to social exclusion analysis (e.g., by examining the benefit of employment accessibility for

disadvantaged populations before and after the implementation of a transport scheme), economic valuation of accessibility effects (e.g., in cost-benefit analyses and studies assessing the impact of changes in public transport accessibility on house prices) and behavior analysis vis-a-vis accessibility measures (e.g., walking behavior dependence on different residential neighborhood accessibility qualities). It has also been used to explore questions of oil dependence, climate change and other concerns (Curtis et al., 2013).

35.3 The Measure of Potential Accessibility

Today, methods to assess accessibility quality are often used in superordinate planning procedures, like regional transport planning, where a central goal is to provide citizens with a certain level of access to various services. For instance, the approach used by Germany's agency responsible for regional planning calculates travel times to major service facilities, like airports or hospitals (Bundesinstitut für Bau-, Stadt- und Raumforschung, accessed March 2015). The results, typically visualized by multi-colored maps, give useful insights into population access to certain services, thus aiding transport infrastructure planning. In this approach, travel times are calculated to a *next* airport, *next* hospital and *next* autobahn access; thus, the implicit assumption is that citizens' needs are fulfilled by one (i.e., the *next*, or closest in terms of travel times) type of facility.

An accessibility measure becomes significant, however, if not just the ability to reach *the nearest* facility serving a particular need is taken into account, but also a *set of multiple reachable* facilities serving the same need; different facilities of the same type may offer varying qualities of a given service. Services may also expand and improve when combined with complementary services provided by another facilities of the same type. For instance, a person planning to take a holiday trip by plane will probably consider several airports in his/her vicinity, instead of just looking at flights offered from the nearest airport. Thus, accessibility to airports should be made dependent on the ability to reach all local airports instead of just the nearest one. Facilities offering medical services may serve as another example. Considering the nearest hospital may be sufficient when looking at simple services like first aid, presumably available at almost *any* hospital. In other cases, however, medical services accessibility should consider several hospitals in the vicinity because they are likely to offer different specialized medical treatment. Consideration of a set of multiple facilities, potentially useful from the perspective of a person at a given location, corresponds to taking into account the **land-use component** of accessibility defined above.

Hansen (1959) considers the whole scope of potential activity facilities, where an accessibility measure *potential accessibility* is defined. Such measures of potential accessibility are specified as the (weighted) sum over the accessibilities of several specific activity facilities (e.g., shopping, leisure etc.) and take the mathematical form

$$A_{\ell} = g\left(\sum_j a_j f(c_{\ell j})\right), \quad (35.1)$$

where j are all possible destinations (opportunities), a_j describes opportunity attractiveness, $c_{\ell j}$ denotes the generalized traveling cost between origin ℓ and destination j , $f(c)$ is an impedance function which (typically) decreases with increasing distance and $g(\cdot)$ denotes an arbitrary, but usually monotonically increasing function. The weight of each opportunity j is thus the product of the destination's attractiveness, a_j , and the ease of getting there, $f(c_{\ell j})$. As seen in its functional form, this type of accessibility measure is related to gravity models used in trip generation models, explaining why this measure is sometimes also referred to as a "gravity type" accessibility indicator (Morris et al., 1979). The (quantitative) accessibility measure used in the MATSim accessibility

extension is expressed in this mathematical form and may thus be seen as a *potential accessibility* measure.

It is important to note that the above-defined measure quantifies how accessible a given location ℓ is to certain services j . This kind of accessibility is *outgoing accessibility*, while a measure of *incoming accessibility* quantifies how accessible a given destination location j is *from* other locations. Nicolai and Nagel (2014) discuss circumstances under which these measures are interchangeable.

35.4 Accessibility Computation Integrated with Transport Simulation

As mentioned above, accessibility computations are often based on travel times (Bundesinstitut für Bau-, Stadt- und Raumforschung, accessed March 2015; Büttner et al., 2010), which serve as an impedance measure. Ways of calculating these travel times can, however, vary significantly. The simplest way to calculate a travel time between two locations is to measure the Euclidean distance (beeline distance) between these two locations and multiply with some average speed. According to Geurs and van Wee (2004), this is the usual approach in location-, person-, and utility-based accessibility approaches, where the focus is not specifically on the transport system.

To strengthen the **transport component** of accessibility (as introduced above) and make accessibility measure sensitive to transport infrastructure changes, a better representation of the travel impedance between origins and destinations is required. The most common approach is travel time calculation using shortest-path algorithms on a real-world transport infrastructure network representation. Many accessibility computations are embedded into GIS software, offering procedures for network-based computations (Bundesinstitut für Bau-, Stadt- und Raumforschung, accessed March 2015; Curtis et al., 2013; Büttner et al., 2010).

The accessibility extension in MATSim also offers this type of accessibility computation. To run it, an accessibility controller listener, e.g., the `GridBasedAccessibilityControllerListenerV3` must be added to the MATSim controller. An example is given in `RunAccessibilityExample` (see <http://matsim.org/javadoc> → accessibility → `RunAccessibilityExample` for details). As input, a network file and a facilities file are required (for more information on networks and facilities, refer to Section 4.1.1 and Section 6.4 of this book). This procedure is more disaggregate than many common approaches to accessibility computations, where single facilities are seldom considered; there, structural data like zone sizes, number of jobs, or total sales area are used to represent the *potential* of a given zone (Büttner et al., 2010; Gulhan et al., 2014) (also see Section 35.6).

Either way, performing an accessibility computation this way can be regarded as a *supply-based approach*, since both supply with transport infrastructure (required to reach a given location) and supply with activity opportunities at these locations are taken into account. The utilization of these two supply dimension by users, i.e., the dimension of *demand* is, however, not considered in this approach. Therefore, no *effects of competition* (Geurs and van Wee, 2004), either for transport infrastructure resources (defined by network capacities), or activity facilities capacities, are taken into account. It is obvious, however, that supply and demand interaction effects are relevant, because opportunities may disappear if they can no longer be reached within reasonable travel times, or when activity facility capacities are exceeded.

By considering demand-supply interaction effects in addition to just the supply side, the scope of the accessibility calculation can be significantly increased. Gauging these effects on *facility capacities* can be addressed by specifying facility capacities in the according value in the `facilities` input file. Observation of *network capacities* and their effects on agents' behavior is one of the core features of the MATSim transport simulation. This is also one major argument for the integration of an accessibility computation with the dynamic transport simulation system MATSim. While other accessibility tools—the majority based on GIS systems (Bundesinstitut für Bau-, Stadt- und Raumforschung, accessed March 2015; Curtis et al., 2013; Büttner et al., 2010; Liu and Zhu, 2004;

Gulhan et al., 2014)—can calculate travel times on a routed network, they do not calculate accessibilities dependent on transport infrastructure usage level. This property, is, however, essential when making accessibility measures sensitive to transport demand management policies, i.e., transport system changes that do not alter the transport infrastructure and are thus not captured by models considering only the supply side.

To take these effects into account, the MATSim accessibility extension must be run with a MATSim transport simulation. To do so, an initial plans file (as described in Chapter 2 of this book) needs to be specified in the MATSim config file. Furthermore, the value `timeOfDay` in the accessibility module of the MATSim config file needs to be specified. If then, as described, an accessibility controller listener is added to the MATSim controller, the best-path travel times, on which the accessibility computation will be performed, are taken from travel times observed in the MATSim transport simulation at the time specified by the value `timeOfDay`. This is useful when transport demand level varies significantly during the day; for instance, with morning and afternoon peaks; it also allows transport policy accessibility changes (and decision makers' reactions) to be better analyzed.

35.5 Econometric Interpretation

As pointed out by Morris et al. (1979), accessibility indicators provide a very useful way to summarize a large volume of information on household locations and how they relate to urban activities' distribution and connecting transport systems. They also take land use, the transport system and their inter-dependencies into account holistically. Curtis et al. (2013) explain that accessibility assessment tools overcome policy innovation restrictions associated with traditional transport planning practice, pointing out that use of such tools enables examination of a broader range of policy issues.

For effective policy decisions, accessibility assessment tools must be economically interpretable. To make an accessibility measure clearest in an econometric evaluation (e.g., cost-benefit analyses), it seems sensible to adapt equation 35.1 as follows: $g(\cdot) = \ln(\cdot)$, $a_j = 1$, $f(c_{\ell j}) = e^{-c_{\ell j}}$, and $-c_{\ell j} = V_{\ell j}$. Thus, equation 35.1 becomes

$$A_{\ell} := \ln \sum_k e^{V_{\ell k}}, \quad (35.2)$$

where k denotes all possible destinations and $V_{\ell k}$ equals the disutility of traveling from location ℓ to destination k . Equation (35.2) is the so-called logsum term of exponentials and can be interpreted as the expected maximum utility (e.g., Ben-Akiva and Lerman, 1985; de Jong et al., 2007). Equation 35.2 can be derived by assuming that the full utility of destination location k as perceived at origin location ℓ , is $U_{\ell k} = V_{base} + V_{\ell k} + \epsilon_{\ell k}$, where V_{base} is a base utility for performing a given activity without considering its location, $V_{\ell k}$ is the systematic or observed disutility of traveling to from origin ℓ to destination k , and $\epsilon_{\ell k}$ is a random term which absorbs the randomness of the disutility of traveling, as well as fluctuations in utility around V_{base} . Under the usual assumption that the $\epsilon_{\ell k}$ are independent and identically (iid) Gumbel-distributed random variables, the expectation value of $U_{\ell k}$ becomes

$$E(U_{\ell}) = E(\max_k U_{\ell k}) = \ln \sum_k e^{V_{\ell k}} + Const \equiv A_{\ell} + Const. \quad (35.3)$$

$Const$ does not need to be considered, as it is invariant for all locations. As a consequence of dropping the positive $Const$, A_{ℓ} may take negative values.

Geurs et al. (2012a), for instance, use the logsum measure of user benefits as an alternative to the travel time savings method (i.e., rule-of-half measure) in a case study examining the effects of spatial planning on accessibility benefits and economic efficiency of public transport projects.

35.6 Spatial Resolution, Data, and Computational Aspects

In contrast to many other transport simulations, MATSim is based on coordinates (see Chapter 2 of this book), not zone-based. Therefore, accessibility computation in MATSim can also be conducted independent from any zoning system and, instead, be based on a raster with arbitrary granularity, i.e., adjustable grid size. Depending on the calculation planned (zone-based or grid-based), a `ZoneBasedAccessibilityControlerListenerV3`, or a `GridBasedAccessibilityControlerListenerV3`, respectively, need to be added to the MATSim controller. Unlike the MATSim accessibility extension, most other accessibility assessment tools rely on the zone-based approach (Curtis et al., 2013; Liu and Zhu, 2004; Büttner et al., 2010). More detail about the interpretation of cell- and zone-based accessibility measures is given by Nicolai and Nagel (2014).

Running a grid-based calculation, especially if a high spatial resolution is selected, avoids several issues that could arise (like “self-potential”) if accessibility computations are based on zones (see, e.g., Nicolai and Nagel, 2014). A zone-based approach also makes the measure dependent on size and shape of the geographical units (cf. MAUP (Modifiable Areal Unit Problem)). Due to its typically lower resolution level, a zone-based approach may also not adequately represent local details (Kwan, 1998). This is especially relevant when lower-speed mode accessibilities (like walking) must be considered.

The MATSim accessibility calculation does not require typical zone-based statistical data. Instead, the calculation can be conducted on the basis of so-called VGI (Voluntary Geographic Information) like OSM, which contains activity facilities data on a coordinate-based level. Hence, no reference to any zoning system is necessary when using these data. Furthermore, data from OSM is publicly and freely available; the amount of these data are steadily increasing and quality is improving. In particular, OSM seems to have established itself as a uniform and globally-accessible standard for crowd-sourced and other geo-data, which makes the MATSim accessibility assessment highly portable.

If the coordinate-based (= grid-based = raster-based = cell-based) version of the MATSim accessibility computation is selected, its results can be interpreted as an accessibility field, i.e., as a measure that varies continuously in space. This *accessibility field*, can be visualized by calculating the values on regular grid points. Figure 35.1 gives an example of such a visualization and depicts the accessibility of work places in Nelson Mandela Bay Municipality in South Africa, as calculated by the grid-based MATSim accessibility computation with a grid size of 1 000 meters.

To calculate the accessibility A_ℓ of a given origin location ℓ to opportunity locations k , both the origin location ℓ , and opportunity locations k , are assigned to a road network. If the option to integrate the accessibility computation with the transport simulation, as described in Section 35.4, is chosen, a congested network with time-dependent travel times (as they have been simulated in MATSim) is used. For every ℓ , a so-called *least cost path tree* computation (Lefebvre and Balmer, 2007) is carried out. Accessibility of the same location at a different time of day will usually be different, since congestion patterns vary. The least cost path tree computation determines the best route and the least negative travel utility $V_{\ell k}$ from the origin location ℓ to each opportunity location k , based on Dijkstra’s shortest path algorithm (Dijkstra, 1959). Once the least cost path tree has explored all nodes, the resulting disutilities $V_{\ell k}$ for all opportunities k are queried and the accessibility is calculated, as stated in Equation (35.2) (Nicolai and Nagel, 2014). A crucial question is how to choose the point, i.e., the coordinate, where the accessibility computation is anchored. Most quantitative accessibility tools use geographical centroids of given zones. This is also true when the zone-based MATSim accessibility computation is selected. Alternative ways to select a centroid (e.g., land-use-based centroids; Büttner et al., 2010) are discussed as well. If the grid-based MATSim accessibility computation is selected, the question of choosing a representative point for a spatial zone becomes less relevant, as cells are usually not selected to be as large.

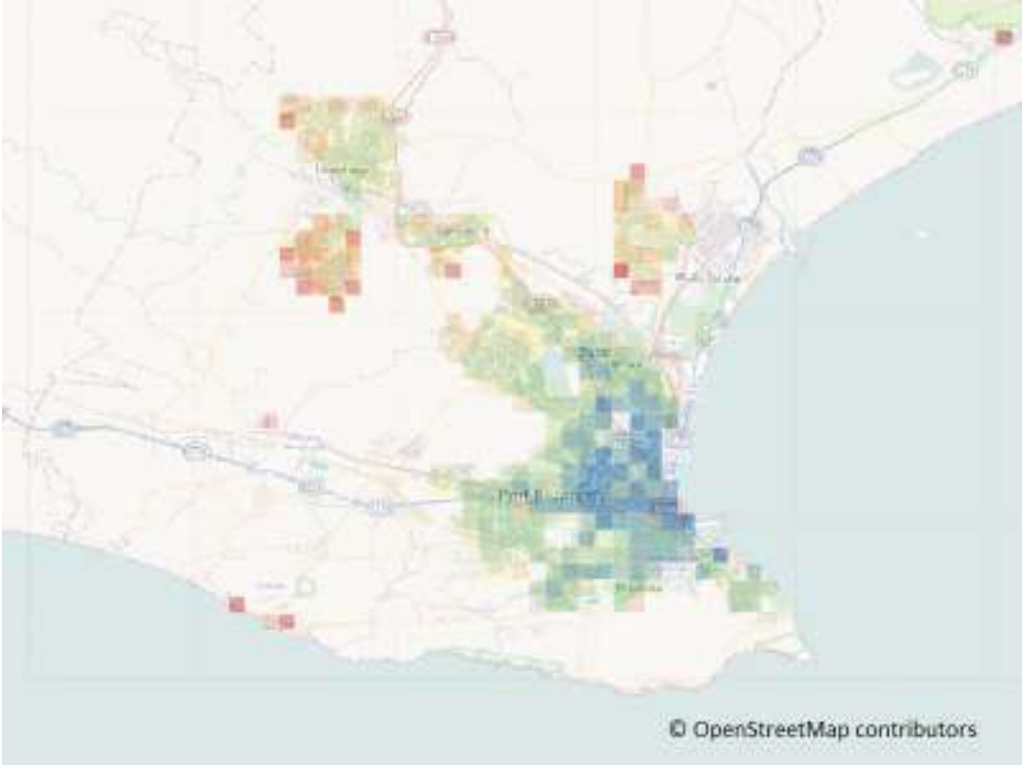


Figure 35.1: Accessibility of work places in Nelson Mandela Bay Municipality calculated by the grid-based MATSim accessibility computation

If the granularity of the grid-based MATSim accessibility computation is increased, origin locations ℓ and opportunity locations k , possibly located off the network, become increasingly important. To keep the approach consistent, the $V_{\ell k}$ calculation has to include disutility of travel to overcome the gap between locations and the road network. Therefore, the disutility of travel calculated by running the least cost path tree computation on the network has to be supplemented by the disutility to access the network from the origin ℓ (network access) and the disutility to access the destination k from the network (network egress). For origin locations ℓ , shortest distance to the network is given either by the Euclidean distance to the nearest node, or the orthogonal distance to the nearest link on the network. For destination locations k , the Euclidean distance to the nearest node is used to determine the shortest distance to the network.

This assumption (i.e., that opportunity locations are attached to the nearest network *node* rather than the nearest network *element*) is, in fact, the only approximation that the MATSim accessibility extension makes for the spatial resolution of opportunities (Nicolai and Nagel, 2014). While this assumption is unlikely to significantly alter accessibility results, it offers great potential for the optimization of computational performance, which has often been a major obstacle to higher-resolved accessibility computations (Kwan, 1998; Büttner et al., 2010). In the concrete case of the MATSim accessibility computation, exploration of the entire network by the least cost path tree is a computationally expensive task.

Thanks to the assumption, it is enough to sum over all opportunities k attached to a node j only once. The travel disutility $V_{\ell k}$ can be deconstructed as

$$V_{\ell k} = V_{\ell j} + V_{jk} \quad \forall k \in j, \quad (35.4)$$

where $k \in j$ denotes all opportunities k attached to node j ,

$$\sum_{k \in j} e^{V_{\ell k}} = \sum_{k \in j} e^{(V_{\ell j} + V_{jk})} = \sum_{k \in j} e^{V_{\ell j}} e^{V_{jk}} = e^{V_{\ell j}} \sum_{k \in j} e^{V_{jk}} =: e^{V_{\ell j}} \cdot Opp_j. \quad (35.5)$$

It is thus sufficient to compute Opp_j once for every network node j , and compute accessibilities as

$$A_{\ell} = \ln \sum_k e^{V_{\ell k}} = \ln \left[\sum_j e^{V_{\ell j}} \cdot Opp_j \right]. \quad (35.6)$$

Therefore, the loop performing the calculation does not have to run over all opportunities k , just over all network nodes j .

Similarly, for each origin location ℓ , the nearest road network node is identified. Locations ℓ that share the same nearest node have different travel disutilities to reach that node, but from then on have the same travel disutility to any other network node j . Exactly like the destinations, the least cost path tree is executed only once and calculated disutilities on the network are reused for all origins ℓ that are mapped on the same nearest network node. Therefore, only the calculation of the network access disutility needs to be performed individually for each origin ℓ . Nicolai and Nagel (2014) show that, due to this run time optimization, computation time increases sub-linearly with resolution. At the same time, they find that no significant further insights can be gained by increasing the resolution beyond a grid resolution of 100 meters.

The application example `RunAccessibilityExample` (see <http://matsim.org/javadoc> → accessibility) performs multiple accessibility computations for different types of activity facilities (e.g., accessibility of workplaces or accessibility of leisure facilities) by adding multiple instances of `GridBasedAccessibilityControlerListenerV3` to the MATSim controller. Other ways of performing distinct accessibility assessments for parts of the land-use system are just as feasible. Figure 35.1 is an example of work place accessibilities.

35.7 Conclusion

There are many different approaches to calculating accessibilities; most focus on a particular component of accessibility, while other components influencing accessibility are represented only in a limited way. Accessibility computations used in transport planning, for instance, represent transport networks, and thus the transport component of accessibility very well, while they usually do not represent facility properties or temporal effects. As pointed out by Geurs and van Wee (2004), it would be optimal if an accessibility computation considered all accessibility components (i.e., transport, land-use, temporal, and the individual component) well. The accessibility extension of MATSim could be an approach to achieve this.

First, transport system dynamics are represented by the accessibility computation integration with the MATSim dynamic traffic simulation. Second, land use is represented in a very disaggregate way; single facilities' locations and attributes are taken into account. Third, the temporal dimension can be observed by representing facilities' opening times and time-dependent travel times on the network; these are given as a MATSim dynamic traffic simulation output. Finally, individual characteristics can be taken into account; in the MATSim simulation, each individual is represented by its own software object, i.e., an agent, whose properties could be considered in the accessibilities calculation.

Actual accessibility values calculated by the MATSim accessibility extension take the form of *potential accessibility measure*, as originally defined by Hansen (1959). The specific selection of the measure's mathematical form allows results to be interpreted as logsum values, making them

suitable for utilization in economic evaluations like benefit-cost analyses. Because the MATSim accessibility extension can rely solely on publicly and freely available data, e.g., data from OSM, it is highly portable. By distinguishing activity facilities along various potential dimensions, many different analyses can be conducted. In the code example given (see <http://matsim.org/javadoc> → accessibility → `RunAccessibilityExample`), for instance, accessibilities for different land uses, i.e., different types of activity opportunities, are calculated. Being grid- instead of zone-based (which most other accessibility tools are), avoids certain problems associated with zones. At the same time, computations are still within reasonable ranges, partly due to a runtime optimization that reuses computational steps for locations sharing the nearest network node.

CHAPTER 36

Emission Modeling

Benjamin Kickhöfer

36.1 Basic Information

Entry point to documentation:

<http://matsim.org/extensions> → emissions

Invoking the module:

<http://matsim.org/javadoc> → emissions → `RunEmissionToolOnlineExample` class,
`RunEmissionToolOfflineExample` class

Selected publications:

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

36.2 Introduction

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

The chapter is structured as follows: Section 36.3 reviews literature for other attempts to model transport-related emissions. Section 36.4 presents an overview of the “EMT (Emission Modeling Tool, see Chapter 36)” and Section 36.5 shows how the tool is embedded in MATSim’s software structure.

How to cite this book chapter:

Kickhöfer, B. 2016. Emission Modeling. In: Horni, A, Nagel, K and Axhausen, K W. (eds.) *The Multi-Agent Transport Simulation MATSim*, Pp. 247–252. London: Ubiquity Press. DOI: <http://dx.doi.org/10.5334/baw.36>. License: CC-BY 4.0

36.3 Integrated Approaches for Modeling Transport and Emissions

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

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

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

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

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

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

¹ The PHEM (Passenger Car and Heavy-duty Emission Model) model uses speed trajectories as input and was tested against the output of the EMT by Hülsmann et al. (2011).

Hatzopoulou and Miller (2010) develop a methodology for calculating exhaust emissions, using MATSim as transport model. The approach is therefore similar to the EMT. In contrast to that study, the EMT does not assume fixed exhaust emissions per time unit. It uses a more detailed calculation of emissions based on the two different traffic states: “free flow” and “stop&go”. It is, thus, able to capture congestion effects that emerge, as well as the time spent in traffic jam. Furthermore, the EMT calculates exhaust emissions for passenger cars *and* for trucks. Finally, since the methodology is based on HBEFA, it can be transferred to any scenario in Europe.

36.4 Emission Calculation

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

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

The calculation of warm emissions is composed of two steps:

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

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

$$l_s = \frac{l v_s (v_f - v)}{v(v_f - v_s)}, \quad (36.1)$$

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

where l is the link length in kilometers from the network, v_s is the stop&go speed in km/h for the HBEFA road type, v_f is the free flow speed in km/h from the network, and $v = \frac{l}{t}$ is the average speed on the link for the vehicle, t being the link travel time of the vehicle in the simulation. For the derivation of Equation (36.1), please refer to Kickhöfer (2014). The distance that the car is driving in free flow traffic state is then simply the remaining link length $l_f = l - l_s$. The interpretation of this approach: Cars drive in free flow until they have to wait in a queue. Stop&go traffic state applies only in the queue. According to the MATSim queue model presented in Section 1.3, a queue emerges if demand exceeds capacity of a road segment, which can also result in spill-back effects on upstream road segments. The length of the queue is, thus, approximated by Equation 36.1, where the average speed v on a link is the only exogenous variable.

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

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

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

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

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

36.5 Software Structure

The information in this section refers to code that can be found in the MATSim repository. In the following, the software structure of the EMT at revision 30 058 is described. For information on how to use the tool, please use the entry points listed at the beginning of this Chapter 36.

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

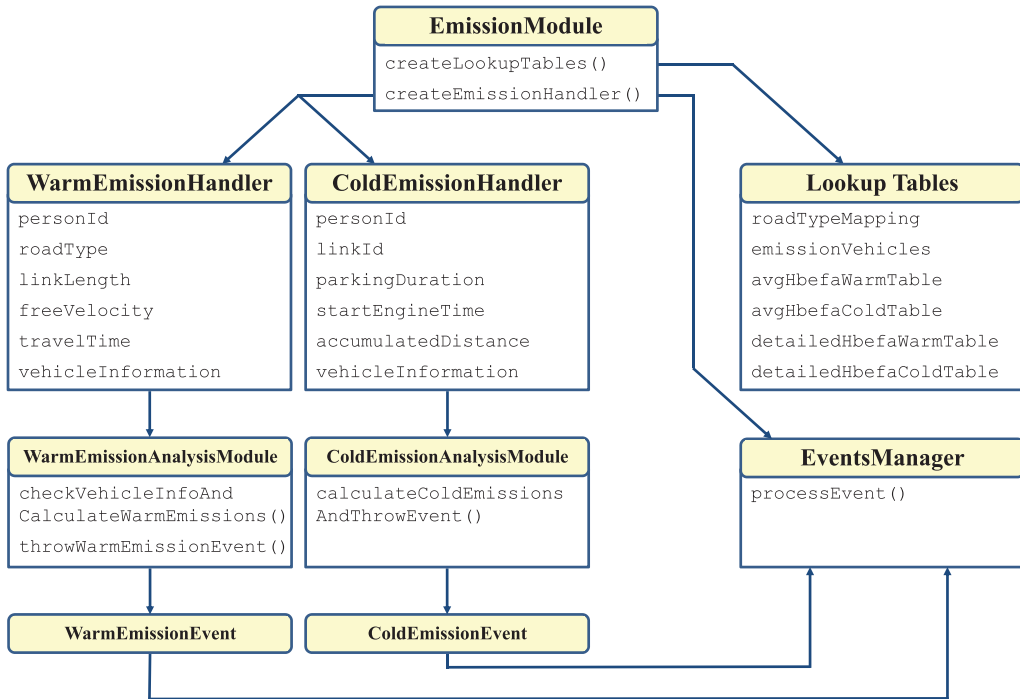


Figure 36.1: Software structure of the emission modeling tool.

Figure 36.1 shows the simplified software structure of the EMT. The core of the tool is the `EmissionModule` which needs to be created before the simulation starts. There are also two public methods that must be called: `createLookupTables()` and `createEmissionHandler()`.

The former creates lookup tables from input data that has to be exported from the HBEFA database. The path to these input files can be configured in the `EmissionsConfigGroup`. Mandatory input are files for the creation of `roadTypeMapping`, `emissionVehicles`, `avgHbfaWarmTable`, and `avgHbfaColdTable`. The first lookup table maps road types from the MATSim network to HBEFA road types. For this mapping, it is necessary to classify the network road types into HBEFA categories; this requires some transport engineering knowledge. The second lookup table defines the vehicle attributes of every owner in the population. It should therefore be generated during the population synthesis process. If no detailed information is available, the vehicle lookup table still needs to specify whether the vehicle is a car or a truck. The current implementation uses the MATSim vehicle interface `Vehicles` as container for storing the relevant data in `VehicleType`.⁴ The last two mandatory lookup tables (`avgHbfaWarmTable` and `avgHbfaColdTable`) provide warm and cold emission factors in g/km , respectively. The data is stored using a unique key. For the construction of this key, information from `roadTypeMapping` and `emissionVehicles` is needed, as well as information derived from the simulation as described in Section 36.4. The latter information is depicted in Figure 36.1 as variables of the two classes `WarmEmissionHandler` and `ColdEmissionHandler`. These two handlers implement several MATSim `EventHandler` interfaces to extract necessary information from the simulation. After gathering this information, the `WarmEmissionHandler` asks its `WarmEmissionAnalysisModule` to reconstruct the key and look up the emission factors in the

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

respective table. Similarly, the `ColdEmissionHandler` asks the `ColdEmissionAnalysisModule`. These analysis modules then create `Warm/ColdEmissionEvents`, which follow the `MATSim Event` interface definition. Finally, the resulting events stream is written in a joint emission events file by a separate `EventsManager`.

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

CHAPTER 37

Interactive Analysis and Decision Support with MATSim

Alexander Erath and Pieter Fourie

37.1 Basic Information

Entry point to documentation:

<http://matsim.org/extensions> → `travelsummary`

Invoking the module:

<http://matsim.org/javadoc> → `travelsummary` → `RunEventsToTravelDiaries`

Selected publications:

This chapter is largely based on work in Erath et al. (2013), where the interested reader will find references for further reading.

37.2 Introduction

Agent-Based Simulation Means Lots of Data Agent-based transport demand models require managing and integrating data sources several orders of magnitude larger than traditional aggregate models. In a truly disaggregate demand description, as seen in our MATSim implementation for Singapore, spatial data represents individual buildings and land parcels, not zones; travel demand takes the form of a full activity diary with connecting trips for every individual, based on their personal demographic attributes, instead of an aggregate number of trips from zone to zone for a specific time period. For this reason, input data for an aggregate four-step (or related) demand model can generally be edited on a laptop, using standard spreadsheet software, whereas agent-based modeling requires the manipulation and synthesis of large stores of structured, hierarchical data, frequently exceeding most personal computer capacity.

How to cite this book chapter:

Erath, A and Fourie, P. 2016. Interactive Analysis and Decision Support with MATSim. In: Horni, A, Nagel, K and Axhausen, K W. (eds.) *The Multi-Agent Transport Simulation MATSim*, Pp. 253–258. London: Ubiquity Press. DOI: <http://dx.doi.org/10.5334/baw.37>. License: CC-BY 4.0

How MATSim Stores Data MATSim stores and retrieves data from XML, because XML reflects objects' hierarchical structure in the simulation and is readable. However, performing general exploratory analysis of large XML data stores is usually poorly supported by most data analysis software packages, especially GIS-based systems. To perform analyses, expert knowledge of XML querying technologies like XPath and XQuery is required (or Java, if one performs more specialized analysis on the objects themselves). In our experience, this specialized knowledge is lacking in transport and urban spatial planning practice. Therefore, in most MATSim applications so far, authorities, and other interested parties, must formulate their desired analysis in advance and have expert consultants perform the analysis. Any queries resulting from the analysis require another consultation cycle and the client's perceived value declines, due to both lack of interactivity and model ownership feeling. We believe this lack of a broadly supported exploratory data analysis interface, and the customer experience the interface can create, presents a considerable barrier to entry for many authorities and operators interested in using MATSim.

How Customers Interact With Data: Relational Databases, GUI-Driven Interaction Most transport and urban spatial planning customers rely on mature, GUI-driven software, such as ArcGIS (ESRI, 2011), EMME/3 (INRO, 2015), the PTV (PTV, 2009) transport planning suite, or even Microsoft Excel; all of these connect to relational databases and perform queries on large data sets. Many analysts can explicitly query databases using the SQL (Structured Query Language); the ODBC (Open Database Connectivity) standard allows software to connect to any relational database regardless of the actual technology driving it. Importantly, many interactive exploratory data analysis software suites, like Tableau, Tibco Spotfire, SAS and the open source R project, support relational databases and ODBC.

37.3 Requirements of a Decision Support Interface to MATSim

The event stream produced by the MATSim mobility simulation represents the transport simulation process at the atomic level. It could be fed into a relational database; an analyst fluent in procedural languages could process it in arbitrary ways. But we expect more general use case scenarios, where most analysts will perform general tasks that can be standardized. To this end, we set about compiling requirements specifications for potential audiences and their use case scenarios, to come up with a general interactive analysis framework and decision support to satisfy most requirements. We developed a set of Java classes to process MATSim input and output, producing tables in a relational database, and an entity relationship diagram that should be intuitive and useful to a large user audience.

37.3.1 Users

This chapter presents a decision support tool geared to decision makers and researchers in the fields of transport planning and operations, spatial planning and spatial economics and geography. Generally speaking, it should serve professionals interested in mobility and spatial analysis, who understand transport modeling principles, but do not have the expertise to operate an agent-based transport simulation directly. Currently, we envision the following stakeholders and some hypothetical questions for a decision-support system—a non-exhaustive list that, we expect, will grow with time:

Transport planners: How many trips occur where, when and what is the activity purpose? What are the socio-demographic characteristics of people performing these trips and activities?

Urban Planners: What are the temporal usage patterns of buildings and the surrounding neighborhood? What is the flow from public transport stops to surrounding buildings?

Policy-Makers: What are the costs and benefits of a new public transport service? Who are the winners and losers when constructing a new road?

Public Transport Operators: What is the breakdown of specific bus lines' ridership?

Service Industry: Which customers are in catchment areas, separated by mode?

37.3.2 Functional Requirements

The decision support framework should facilitate classic transport appraisal methods, such as cost/benefit analysis and evaluation of transport infrastructure spatial impact and policy measures. The framework should allow any sort of spatial analysis, on the finest granularity level provided by the transport model; usually, individual buildings or parcels, as well as public transport stops and selected links, like count stations or tolled road segments. However, these geographic features should be indexed against transport zones, or other geographic areas of interest, to allow customized results aggregation. Furthermore, it should capture all temporal aspects of the simulation; full temporal dynamics are a crucial part of the agent-based approach.

37.4 General Framework for Decision Support

Figure 37.1 shows the general framework as we envision it: data from various sources feeds into a spatially-enabled database, with all geodata transformed to use the same spatial reference system (ideally, using the same projection used for MATSim coordinates, allowing for simple distance calculations). Simple Java programs using the MATSim API and JDBC (Java Database Connectivity) produce XML input data for MATSim scenarios; events from these scenarios are fed back into the database. Analysts query the database to produce “data cubes”, which are aggregations and queries across many database tables. These are designed for specific purposes, such as calibration and validation, location analysis, winner/loser analysis or other application-specific purposes.

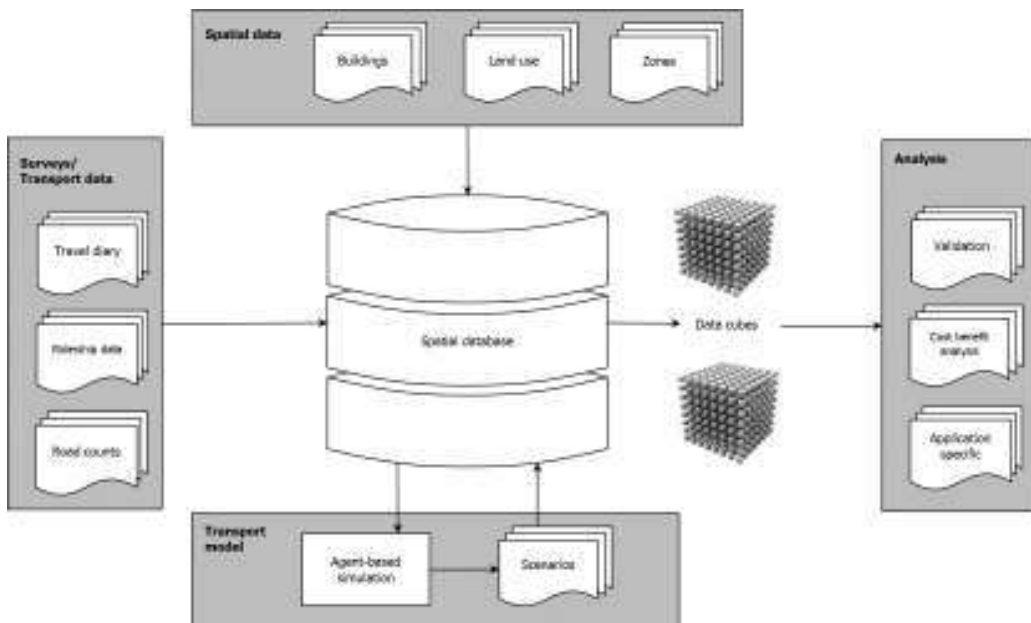


Figure 37.1: General framework of the decision support system.

37.4.1 Entity Relationship Diagram (ERD) for General Purpose Analysis

For entity relationships, we decided that a travel diary format is most suitable for the usual types of analysis, but works especially well for comparison with other data sources when validating simulation output. Most travel surveys take the form of a diary, recording travel time, purpose and mode, as well as aspects of the journey like number of stages, transfer walking and waiting time and in-vehicle time. Routines can be developed to transform survey data and public transport smart card records into the same format with consistent coding. Figure 37.2 shows the ERD (Entity Relationship Diagram) we propose, along with the primary/foreign key relationships between tables that facilitate aggregation and joining of e.g., personal/household attributes, such as income, with travel time experienced in the simulation.

37.4.2 Interactive Analysis Using Business Analytics Software

Modern business analytics software, like Tableau (Tableau Software, 2013), provide interactive data aggregation and visualization from relational databases. While basic analysis of individual tables

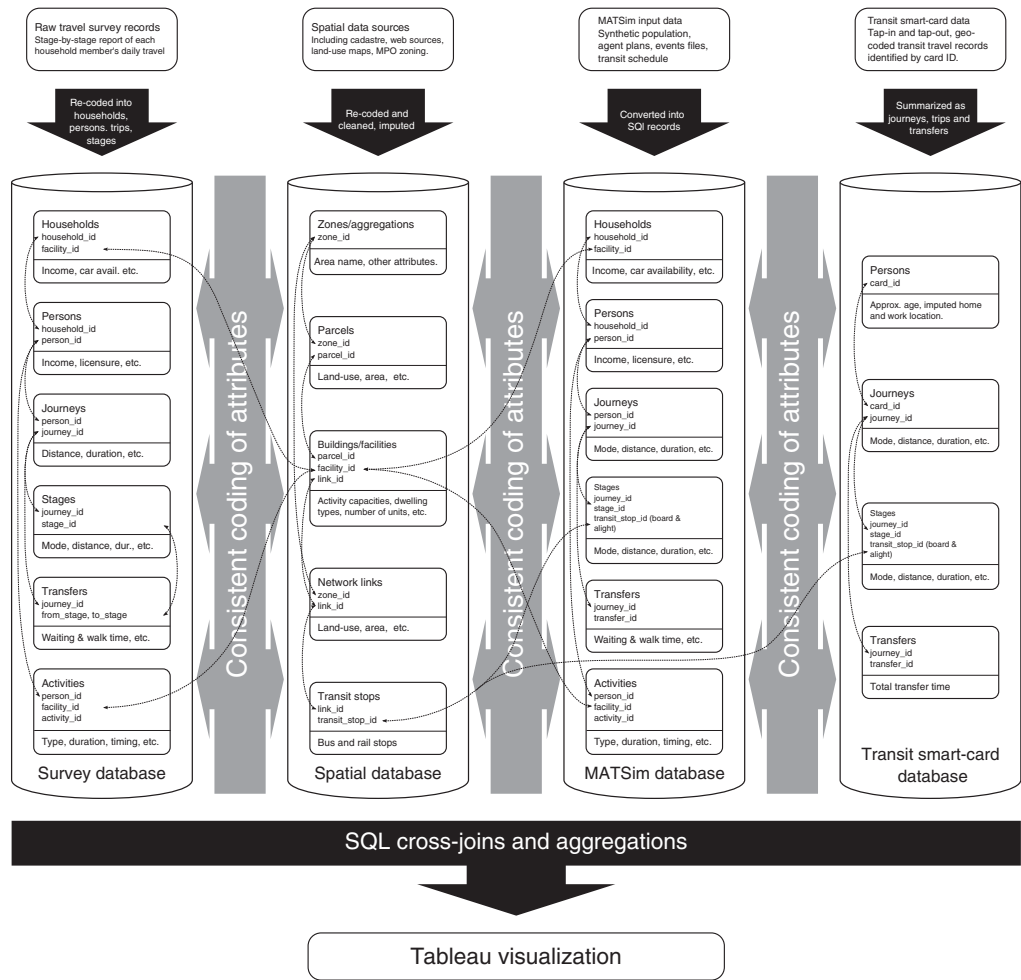


Figure 37.2: Simplified entity relationship diagram showing shared keys across tables.

in our proposed ERD could already provide valuable insight to MATSim simulations, much richer analysis is possible when tapping relationships between different tables in the database. With the help of graphical query building software, little or no knowledge is required to construct SQL scripts that create customized data cubes. These cubes are fed into the business analytics software, which is designed with a relatively programming-agnostic audience in mind. Relying on the familiar paradigm of drag-and-drop interaction in a simple, well-documented GUI, the user constructs “dashboards” summarizing information and allowing interactive aggregation, or drilling-down across multiple dimensions.

Figure 37.3 shows a Tableau visualization comparing public transport ridership from a MATSim simulation to actual smart card data records (transformed into the travel diary format specified in the ERD). Figure 37.4 shows the SQL query used to produce the data frame driving the Tableau analysis. The query exploits the primary/foreign key relationships in the database to perform rapid joins between the different tables.

37.5 Diaries from Events

In the package `contrib.analysis.travelsummary` (Chapter 38), the reader can find a set of classes that will transform their MATSim simulation results into a set of travel diary tables, like those discussed in the preceding section. The package contains a simple GUI class that can be run to specify input data XML files, the location to save output CSV (Comma-Separated Values) files and other information such as a subscript appended to the end of file names to identify different scenarios. These CSV files can be read into a relational database of choice, or directly queried in Tableau, or other interactive analysis software.

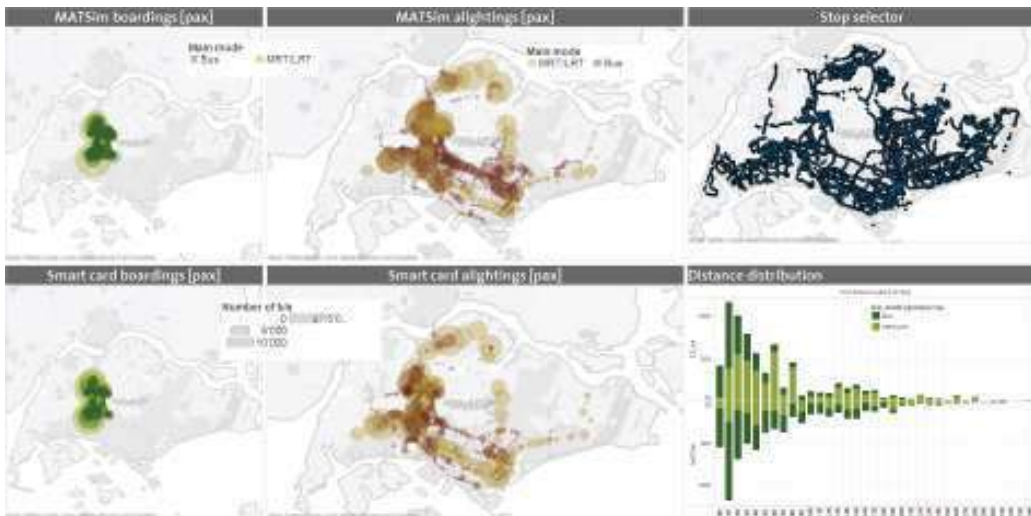


Figure 37.3: Tableau visualization of public transport ridership from a MATSim simulation compared against actual smart card data records in Singapore.

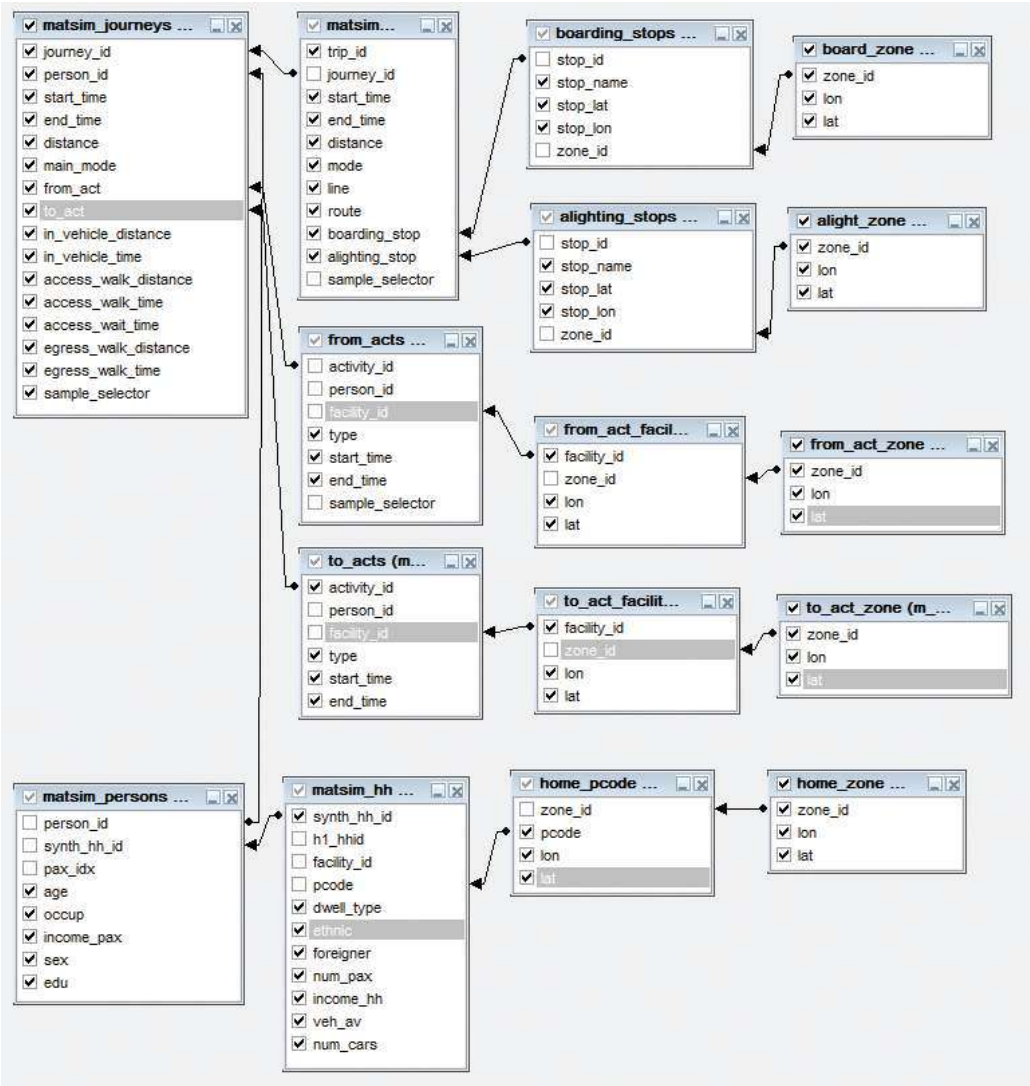


Figure 37.4: A diagram showing how the tables from Figure 37.2 are joined together for visualization in business analytics software, e.g., Tableau, as shown in Figure 37.3.
Source: (Erath et al., 2013)

CHAPTER 38

The “Analysis” Contribution

Kai Nagel

38.1 Basic Information

Entry point to documentation:

<http://matsim.org/extensions> → analysis

Invoking the module:

No standard invocation. See <http://matsim.org/javadoc> → analysis → `RunKNEventsAnalyzer` class for intuition.

Selected publications:

–

38.2 Summary

This contribution collects various analysis tools for MATSim output.

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

How to cite this book chapter:

Nagel, K. 2016. The “Analysis” Contribution. In: Horni, A, Nagel, K and Axhausen, K W. (eds.) *The Multi-Agent Transport Simulation MATSim*, Pp. 259–260. London: Ubiquity Press. DOI: <http://dx.doi.org/10.5334/baw.38>. License: CC-BY 4.0

SUBPART ELEVEN

Computational Performance Improvements

CHAPTER 39

Multi-Modeling in MATSim: PSim

Pieter Fourie

39.1 Basic Information

Entry point to documentation:

<http://matsim.org/extensions> → pseudosimulation

Invoking the module:

<http://matsim.org/javadoc> → pseudosimulation → RunPSim class

Selected publications:

Fourie et al. (2013)

39.2 Introduction

MATSim's major current performance limitation is the network loading simulation, i.e., the mobsim, for example QSim or JDEQSim; this chapter focuses on QSim. As shown earlier, QSim is repeatedly executed in the MATSim loop for the entire agent population (Section 1.2).

With the multi-modeling approach (Fourie et al., 2013), shown in Figure 39.1, a MATSim run periodically replaces QSim for a number of iterations with a simplified meta-model or PSim (Pseudo-Simulation), running approximately one hundred times faster. In risk analysis, these models are called “surrogate models” (Sudret, 2012). PSim uses travel time information from the preceding QSim iteration to estimate how well an agent day plan might perform, allowing multiple iterations of mutation and evaluation between QSim iterations to more rapidly explore the agents' solution space, producing better performing plans in a shorter time.

How to cite this book chapter:

Fourie, P. 2016. Multi-Modeling in MATSim: PSim. In: Horni, A, Nagel, K and Axhausen, K W. (eds.) *The Multi-Agent Transport Simulation MATSim*, Pp. 263–266. London: Ubiquity Press. DOI: <http://dx.doi.org/10.5334/baw.39>. License: CC-BY 4.0

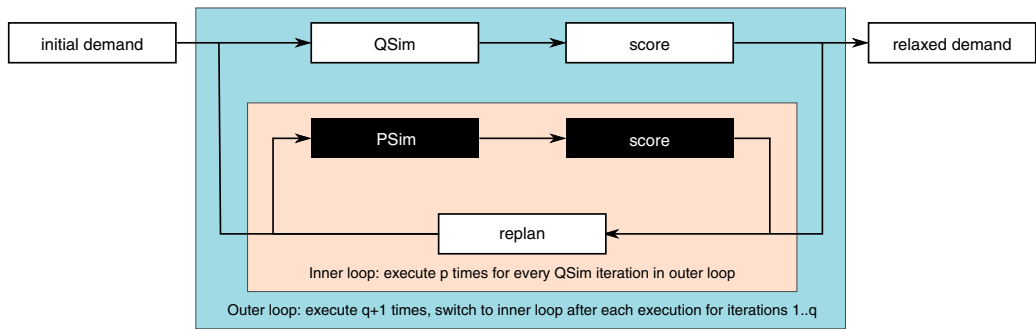


Figure 39.1: Operation of a MATSim run implementing pseudo-simulation.

Source: Fourie et al. (2013), Figure 1, p. 69

39.3 Basic Idea

PSim exploits classes that record various network performance aspects during queue simulations and uses them as an approximate meta-model of QSim. It fires the same sequence of events for car and public transport passengers that is produced during a QSim mobility simulation, except that event timings are approximate values expected at the time of day they occur.

For private vehicle traffic, it calls the

```
getLinkTravelTime(Link link, double time, Person person, Vehicle vehicle)
```

method of classes implementing the `TravelTime` interface to fire `LinkEnterEvents` and `LinkLeaveEvents` at appropriate times for all car route links. For public transportation, the events sequence generated for a passenger traveling on a particular service relies on a meta-model of stop-to-stop travel times (interface `StopStopTime`) and waiting times at stops (interface `WaitTime`); both concepts were developed by Sergio Ordóñez at the Future Cities Laboratory (package `playground.singapore.transitRouterEventsBased`).

PSim plans are scored using the same function as QSim iterations and are compatible with most replanning modules in MATSim. Following a series of PSim iterations, a plan is selected for each agent, in the usual fashion, and a QSim iteration is run to start a new cycle. The various classes used in PSim are updated with the latest network performance information and the process repeats.

39.4 Performance

Initial tests on the Zürich scenario (described in Chapter 56) have shown a dramatic decrease in computation times, compared to the default QSim-only approach; performance improves linearly with an increasing number of computational cores. Figure 39.2 compares the PSim-approach, in two configurations, against the existing approach, for a 10 % sample of private vehicle traffic in Zürich. All simulations were run until they reached a target score, i.e., the score reached after running the standard approach for 100 iterations. The first PSim-implementing configuration uses the same rate of plan mutation as the QSim-only approach, where 30 % of agents are selected for plan mutation (replanning) after each iteration, whether it is a QSim or PSim iteration. The new approach requires fewer QSim iterations to reach a target score, but requires more time for replanning. Replanning is fully multi-threaded, with no synchronization between cores required, so its performance increases linearly, with increasing number of cores; times improve more dramatically with the new approach than the standard approach. In the second configuration, the mutation rate

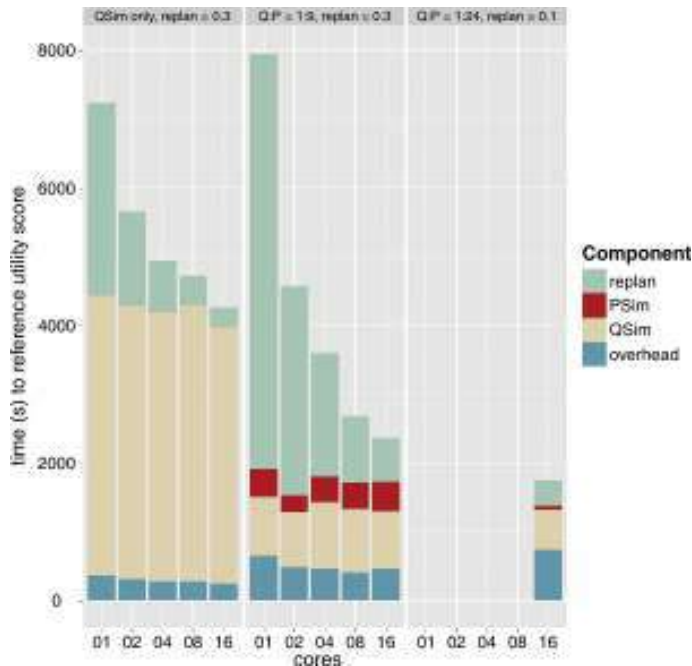


Figure 39.2: Computation time contributions vs. number of computational cores for QSim-only (0.3 replanning rate), 9 PSim iterations per QSim iteration at 0.3 replanning rate, and 24 PSim iterations per QSim iteration at 0.1 replanning rate.

Source: Fourie et al. (2013), Figure 4, p. 73

is reduced and the number of PSim iterations between QSim iterations increased to 24 for each QSim iteration. The system now tests many more combinations of different mutation operations (four in this case: activity timing, mode choice, secondary activity location choice, and re-routing), to reach the target state much faster, even though it produces a smaller expected number of mutated plans per agent between QSim iterations (three for configuration 1, 2.5 plans for configuration 2).

This last point raises an interesting issue; namely, that the distribution of mutation operation numbers can be dramatically spread out with the PSim approach, because increasing the number of iterations is relatively cheap. This should make the approach preferable, especially with random mutation-producing replanning strategies, where a large number of mutations are needed to produce a relaxed simulation state.

For a detailed discussion of the meta-modeling approach and the results of applying this method to the Zürich scenario, refer to Fourie et al. (2013).

39.4.1 Distributed Computing

Because PSim executes plans independently from each other, requiring no coordination of computational processes, it is possible to distribute it across multiple nodes, with no need of shared memory, as illustrated in Figure 39.3. To this end, we (Fourie and Ordóñez, FCL (Future Cities Laboratory)) are implementing a simple messaging protocol to transmit network performance objects to PSim slave nodes from a master node running QSim only. Slave nodes perform replanning operations and evaluate plans in a pre-determined number of PSim iterations per cycle. At the start of each QSim iteration, a single plan for each agent is transmitted back to the master from

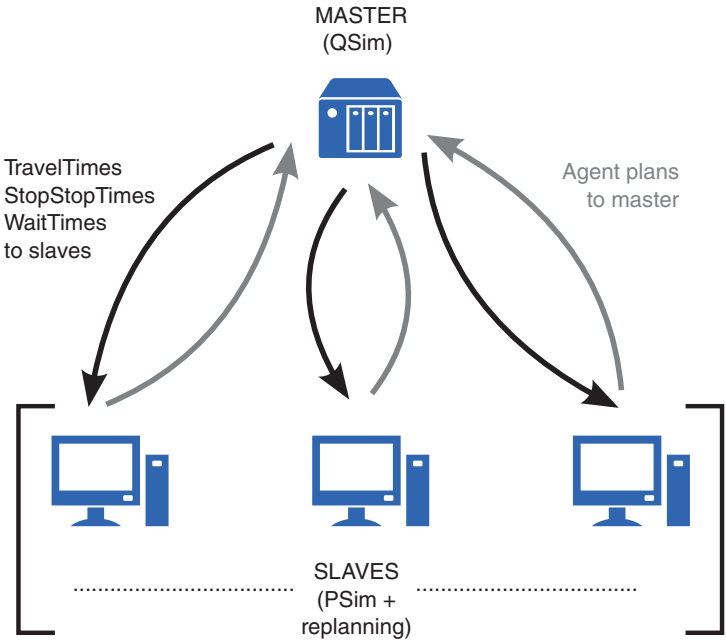


Figure 39.3: Master-slave configuration for running PSim in distributed mode, across many slave computer nodes in a local area network or in a cloud computational framework. The master runs selected plans in a full queue simulation and transmits updated travel time information to slave nodes after every iteration. In turn, slaves produce and evaluate new plans in repeated PSim/replanning cycles, sending the master a single plan for each agent at the start of a QSim iteration.

all the slaves, and updated `TravelTimes`, `StopStopTimes` and `WaitTimes` are rendered during the full mobility simulation, to be transmitted back to the slaves in the next cycle.

The approach yielded promising results, with a reduction in the number of QSim iterations, as in the previous work, as well as the potential for running large-scale simulations on much cheaper hardware than the current approach, that demands expensive shared memory servers. Most importantly, all replanning takes place in parallel with the QSim running on the master, so the time spent waiting for replanning operations can be reduced to nil. This performance increase is especially useful for large scenarios implementing public transportation, where the time spent replanning can be up to twice that of the queue simulation.

Other Experiences with Computational Performance Improvements

Kai Nagel

MATSim has always had the simulation of large regions as its goal, and as such was always interested in high computational performance. The team had, when it started with the Java-based MATSim (cf. 46.2.1.4), considerable experience in parallel computing (Nagel and Schleicher, 1994; Rickert and Nagel, 2001; Nagel and Rickert, 2001; Cetin et al., 2003) as well as with more general message-based approaches (Gloor and Nagel, 2005) that resemble today's Protocol Buffers (Google Developers, 2015). However, the move to Java (see Section 46.2.1.4), a decision for faster conceptual progress and reduced maintenance effort, also had the consequence that the MPI (Message Passing Interface) approach to parallel computing could no longer be used and was thus given up. See Section 46.2.1.4 for details.

The behavioral modules of MATSim, such as route (Section 4.5.1.2) or destination (Chapter 27) innovation, are conceptually straightforward to parallelize by multi-threading, and that was implemented in MATSim from early on (Balmer et al., 2009b, see Section 4.2.3 how to use this). The remaining challenge then is to parallelize the mobsim, in which the parallel threads need to interact closely. For example, assume that we compute 24 hours of traffic in 120 seconds of computing time (cf. Table 40.1). With the 1 second time steps used in the QSim this means 720 update rounds per second, and thus 720 inter-thread interactions per second.

An attempt to use the CUDA (Compute Unified Device Architecture, a parallel computing platform and API by NVIDIA) for the C language (Strippgen and Nagel, 2009b,a; Strippgen, 2009) ran into the same problems as the earlier parallel DEQSim also written in C/C++ (Charypar et al., 2007a): The time necessary to transmit the necessary information back and forth between the Java-based MATSim and the C/C++-based external package used up all the performance gains. In consequence, the DEQSim was re-implemented as the so-called JDEQSim in Java (Waraich et al., 2015, also see Section 4.3.2). Before parallelizing the JDEQSim, however, it was decided to first accelerate the processing of the events since that was identified as the main bottleneck. Section 4.2.3 describes

How to cite this book chapter:

Nagel, K. 2016. Other Experiences with Computational Performance Improvements. In: Horni, A, Nagel, K and Axhausen, K W. (eds.) *The Multi-Agent Transport Simulation MATSim*, Pp. 267–268. London: Ubiquity Press. DOI: <http://dx.doi.org/10.5334/baw.40>. License: CC-BY 4.0

how to use parallel events handling. The parallel version of the JDEQSim (Waraich et al., 2015) never made it into the MATSim main repository.

At the same time, the standard QSim was improved by other people, for example by keeping track of active links and not doing any computation on links without activity. Ch. Dobler made the QSim multi-threaded. He reported (Dobler, 2013, Chapter 5) close-to-linear speed-ups with large scenarios, but only small—if any—performance gains with small scenarios. That is, multi-threading helped greatly with overall computing times for large scenarios on large shared-memory computers, but little with quick turn-around during experimentation. More recent hardware seems to have improved the situation also for small scenarios (Table 40.1) so that it was eventually decided to remove the single-threaded variant of the QSim and concentrate development on the multi-threaded variant only.

Lämmel et al. (2016) experiments with using Protocol Buffers (Protocol Buffers web page, accessed 2015) in order to couple two different mobsims.

The PSim (Chapter 39) addresses the problem from a different angle: Rather than accelerating the QSim itself, it attempts to make use of the fact that (1) adding or removing a small number of synthetic travelers does not change congestion patterns very much and thus alternative plans can be evaluated in parallel, and (2) the congestion patterns generated by the mobsim do not vary that much from one iteration to the next so that the mobsim does not have to be re-run every time after some synthetic travelers have moved to different alternatives.

Märki et al. (2014) and Dobler (2013) point out that the number of iterations to reach equilibrium can be reduced when the synthetic travelers perform within-day re-routing – this points into the same direction as Lu et al. (2015) who claim that equilibrium iterations will not be necessary at all with well-calibrated behavioral models and a realistic starting point.

MATSim needs, at least for large scenarios, a large amount of RAM. One could say that within the usual space-time tradeoff in computation,¹ in most situations MATSim rather consumes more memory in order to reduce the computation time. Memory-saving compressed routes are available as an option in the <plans> section of the config file. MATSim can be seen as an object-oriented database in RAM; attempts to provide a backing by a relational database were not successful when they were tried (Raney and Nagel, 2004, 2006, ; also see Section 46.2.1.3).

To summarize: (1) The behavioral parts of MATSim parallize easily; the main challenge is the mobsim. (2) The main challenge with parallelizing the mobsim is not so much the pure performance improvement, but to achieve this in a way that it remains integrated with the MATSim main development, and at little or no additional maintenance effort.

Computer	population size	1 thread	4 threads	6 threads	8 threads
laptop 2010	1% = 23 500	432 sec	(X)	(X)	(X)
laptop 2014	1% = 23 500	110 sec		57 sec	55 sec
laptop 2014	10% = 235 000			200 sec	

“(X)” means that the laptop was no longer useful for secondary tasks.

Table 40.1: Computing times of the mobsim for the Gauteng scenario (see Chapter 69) with 523 000 links for different computers, different population sizes, and different numbers of threads. “laptop 2010” refers to a high end Mac Pro laptop from 2010, “laptop 2014” refers to a high end Mac Pro laptop from 2014. We can see a speed increase close to a factor of four from 2010 to 2014, and then in 2014 an additional factor of two with multi-threading. These results were shown at several seminars, but never published elsewhere.

¹ See https://en.wikipedia.org/wiki/Space-time_tradeoff.

SUBPART TWELVE

Other Modules

Evacuation Planning: An Integrated Approach

Gregor Lämmel, Christoph Dobler and Hubert Klüpfel

41.1 Basic Information

Entry point to documentation:

<http://matsim.org/extensions> → evacuation

Invoking the module:

<http://matsim.org/javadoc> → evacuation → `RunEvacuationExample` class

Selected publications:

Lämmel (2011); Lämmel et al. (2009)

This chapter presents an integrated approach for performing evacuation simulations with MATSim using the evacuation contribution. The approach comprises all workflow steps for performing an evacuation analysis: i.e., selecting the evacuation area and defining the population, specifying behavioral parameters (i.e., pre-movement time distribution and mode of evacuation—car or pedestrian) and analyzing the simulation output. These steps can all be performed within one graphical user interface. Additionally, two extensions of MATSim for simulating public transport and changing the network during simulation (i.e., network change events) are accessible from the GUI. In this chapter, the steps for performing such an integrated analysis are described and illustrated based on the Hamburg-Wilhelmsburg example. A detailed case study based on this scenario is given in Chapter 71, as well as in Durst et al. (2014); Hugensch (2012).

41.2 Related Work

Simulation of evacuation processes has attracted much attention in recent decades; reasons include increases in frequency and severity of natural hazards jeopardizing various populations

How to cite this book chapter:

Lämmel, G, Dobler, C and Klüpfel, H. 2016. Evacuation Planning: An Integrated Approach. In: Horni, A, Nagel, K and Axhausen, K W. (eds.) *The Multi-Agent Transport Simulation MATSim*, Pp. 271–282. London: Ubiquity Press. DOI: <http://dx.doi.org/10.5334/baw.41>. License: CC-BY 4.0

and regions, as well as (social) disasters (Rodríguez et al., 2006). Another factor is the availability of large-scale, fast simulation models and tools. Lämmel (2011) discusses such a model employed as a contribution to MATSim. Basically, this model implements the same iterative learning approach as that applied to “regular” transport scenarios. In the first instance, the cost function comprises only travel times, albeit a combination of travel time and travel distance; as a cost function has been investigated as well (Lämmel et al., 2009). Artificial agents represent evacuees trying to improve their evacuation plans from iteration to iteration, by creating new evacuation plans more responsive to the evolving situation. A typical simulation run comprises 500–1 000 iterations. The model is applied to a tsunami-related evacuation of the City of Padang in Indonesia (e.g., Taubenböck et al., 2013; Goseberg et al., 2013); some scenario details are discussed in Chapter 76.

Additional evacuation simulation related work in MATSim is presented by Dobler (2013). The main difference to this chapter’s approach is that agents are allowed to adapt their plans spontaneously, using MATSim’s within-day replanning framework (Dobler et al., 2012) (Chapter 30). Based on a behavioral model, agents coordinate their actions on a household level. If a household is, e.g., not complete when the evacuation starts, each member estimates time needed to return home, as well as the time required to leave the actual evacuation area. Then, the household decides whether meeting at home and leaving together is preferable to each member leaving on its own. Since the behavioral model is implemented on an agent, respectively household level, individual attributes such as children present in the household, or availability of a car, can be taken into account. In contrast to regular MATSim simulations, only a single iteration is performed. Since the agents can optimize their plans continuously using real time information, no further replanning is necessary. As a result, agents do not foresee future events like traffic jams caused by people leaving the threatened area.

An independent evacuation scenario, not using the evacuation, is presented in Chapter 60.

The remainder of this chapter is organized as follows: Section 41.3 gives a brief description on how to set up and run evacuation. A short start guide for evacuation is presented in Section 41.4. Obtaining the required input data is discussed in Section 41.5. Detailed instructions on how to use evacuation’s ScenarioManager, running simulations and analysis is given in Section 41.6. This chapter concludes with a brief outlook in Section 41.7.

41.3 Download MATSim and Evacuation

Although the MATSim version 0.6.0-SNAPSHOT is referred to here, the package should also work with later versions of MATSim.

1. Download the current nightly build of MATSim and evacuation from <http://matsim.org/files/builds/>.
2. Unzip the `Matsim_rxxxxx.zip`, `Matsim_libs.zip` and `evacuation-0.6.0-SNAPSHOT-rxxxxx.zip`.
3. Move the `evacuation-0.6.0-SNAPSHOT-rxxxxx.jar` and `libs` folder from the `evacuation-0.6.0-SNAPSHOT-rxxxxx` directory one level up, i.e., to the directory, where `MATSim_rxxxxx.jar` is located.

Test configuration by invoking

```
java -cp evacuation-0.6.0-SNAPSHOT.jar;MATSim_rxxxxx.jar
org.matsim.contrib.evacuation.scenariomanager.ScenarioManager
```

(It is advisable to copy that command to a file `evacuation.bat`—or `evacuation.sh`, if using a Unix-like operation system. One can then run that file instead of typing the command.)

41.4 The Fifteen-Minute Tour

For just a quick impression, the following steps can be performed within a few minutes:

OSM Go to <http://www.openstreetmap.org>, search for the desired place and download a (small) OSM file. Please choose a small area, e.g., 500 meters by 500 meters; this is sufficient to begin and size of the exported area is limited. For larger areas, a direct download from sites like <http://www.geofabrik.de> is preferable (see next section).

Run the ScenarioManager as described in the previous section.

Create a scenario by clicking the leftmost button first and then New. Go to the directory where the designated project should be saved and name the project file (e.g., london.xml or scenario.xml).

Specify the path of the OSM file (by clicking Set next to network) and the output directory. Leave area and population file as it is, evacuation will handle this. **This step must be performed only once. After the scenario-file has been saved, one can open it in the ScenarioManager.**

Sample size Set the sample size to 0.1, using the mouse or the cursor buttons on your keyboard.

Departure Specify the departure time distribution. Plausible values are: normal distribution, μ and σ 600 seconds (10 minutes), earliest 300 seconds, latest 1200 seconds (20 minutes).

Save the scenario file.

Area Switch to the area tab. One can define the circular evacuation area by keeping the left mouse button pressed and defining the center and radius. Do not forget to save changes.

Population Switch to the population tab and define the population (handling is similar to area). Do not forget to save changes.

Convert Switch to the next tab and convert the scenario to MATSim input files by clicking the run button. The MATSim files will be stored in the output directory specified in the beginning.

Run the MATSim simulation by skipping the next two tabs/buttons (road closures and buses) and switching to the simulation tab (with the “M” for MATSim on the computer screen). Click run. This will take a while. **If an output directory (e.g., from a previous run) already exists, it will be renamed.**

Analyze your simulation results by switching to the final tab after the simulation is finished.

41.5 Input Data (any Place and any Size)

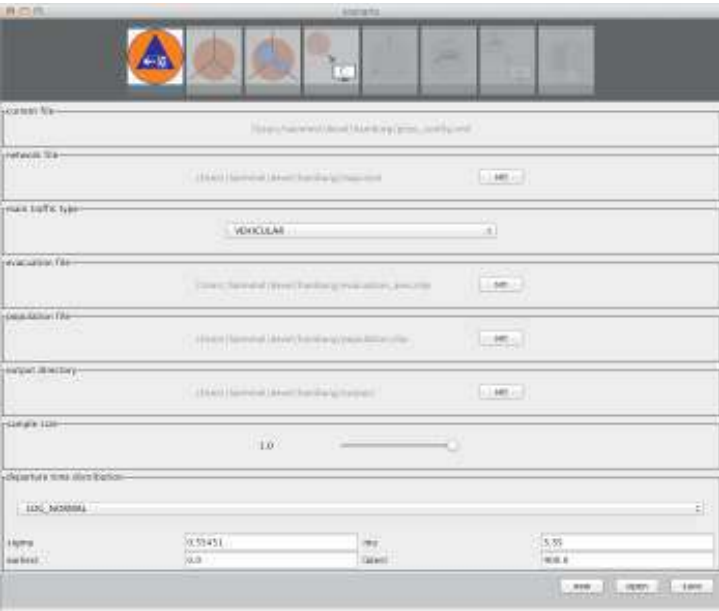
The only external input necessary for performing an evacuation analysis with org.matsim.contrib.evacuation is an OSM file. In this tutorial, we will use the file for Hamburg, Germany. Please go to <http://download.geofabrik.de/europe/germany/hamburg.html> and download the hamburg-latest.osm.bz2 file. This is the only initial preparation needed. Everything else can be done with the ScenarioManager of the GUI.

41.6 Scenario Manager

The scenario setup, evacuation simulation, and analysis are handled by the ScenarioManager from the MATSim contribution package org.matsim.contrib.evacuation.

41.6.1 Scenario Configuration

At startup, the ScenarioManager offers the option to either: define a new scenario configuration or open an existing one from a XML file, which then can be modified. Figure 41.1 shows a screenshot of a scenario configuration in the ScenarioManager and the corresponding XML file, respectively.



(a) ScenarioManager.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<gripa_config xsi:schemaLocation="http://www.matsim.org/files/dtd http
i://matsim.org/files/dtd/gripa_config_v0.1.xsd" xmlns:xsi="http://www.w
3.org/2001/XMLSchema-instance">
  <networkFile>
    <inputFile>/Users/laemmel/devel/hamburg/map.osm</inputFile>
  </networkFile>
  <mainTrafficType>vehicular</mainTrafficType>
  <evacuationAreaFile>
    <inputFile>/Users/laemmel/devel/hamburg/evacuation_area.shp</i
nputFile>
  </evacuationAreaFile>
  <populationFile>
    <inputFile>/Users/laemmel/devel/hamburg/population.shp</inputF
ile>
  </populationFile>
  <outputDir>
    <inputFile>/Users/laemmel/devel/hamburg/output/</inputFile>
  </outputDir>
  <sampleSize>1.0</sampleSize>
  <departureTimeDistribution>
    <distribution>log-normal</distribution>
    <sigma>0.55451</sigma>
    <mu>5.55</mu>
    <earliest>0.0</earliest>
    <latest>900.0</latest>
  </departureTimeDistribution>
</gripa_config>
```

(b) XML file.

Figure 41.1: Illustration of a configuration opened in the ScenarioManager and as XML file.

The evacuation scenario is specified by the following parameters:

- The path to the network file covering the evacuation area: Currently, OSM XML files are supported (*.osm).
- The main traffic type for the simulation: This can either be: VEHICULAR or PEDESTRIAN. Depending on the choice, a vehicular specific (the MATSim default) or a pedestrian-specific

(as discussed in Lämmel et al. (2009); Lämmel (2011)) simulation network will be generated by setting free speed, number of lanes and flow capacity for all links in the network.

- The path to a ESRI shape file describing the extent of the evacuation area, depicted by a simple polygon. This file does not have to be in place right from the beginning; it can be produced manually by the ScenarioManager itself, as discussed later.
- The path to an ESRI shape file detailing the size and distribution of the affected population. This file comprises a set of simple polygons; each polygon has an additional attribute for the number of persons residing at a location inside that polygon. The evacuation area file can be produced with help of the ScenarioManager.
- The path to the output directory where the simulation output and MATSim scenario files will be stored.
- The sample size for the MATSim simulation. A smaller sample size increases the simulation performance, while a larger size might increase accuracy of the results. Typical values are 1.0, 0.1, or 0.01, depending on the scenario and available computing resources.
- Departure time distribution defines the distribution from which departure times for the simulation will be drawn, based on the premise that, in real evacuation situations, all participants probably do not start evacuation simultaneously. People tend to perform pre-evacuation activities before they start, including: picking up relatives, packing food, clothes, valuable belongings, etc. Since number and duration of these activities differs by individual, population departure times are unknown quantities. The ScenarioManager supports three different distributions: (Dirac-delta, normal, and log-normal). If the user chooses the Dirac-delta distribution, then all evacuees will start simultaneously, which might be the worst case (Lämmel and Klüpfel, 2012). By choosing the normal distribution, departure times for individuals are drawn from a normal distribution with mean μ and standard deviation σ , where the parameters μ and σ are given in seconds. As an example, setting $\mu = 1800$ and $\sigma = 900$ will result in a departure time distribution where, on average, after 30 minutes 50 % of the population has departed and 68.3 % of the population departs in time intervals of 30 minutes \pm 15 minutes. If the user chooses log-normal as the distribution, departure times are drawn from a log-normal distribution, where μ and σ are the parameters of the associated normal distribution (a discussion on this matter is given below). The parameters *earliest* and *latest* determine the earliest and latest possible departure time. The normal and log-normal departure time distribution are truncated accordingly.

The departure time distribution is perhaps the most tenuous parameter to set; the authors found no holistic research into this matter. In general, it seems reasonable to assume that many people start evacuating at the same time, or soon after the evacuation order has been issued and as time proceeds, fewer and fewer people are left to depart. This requires a departure time distribution that has a probability density function beginning with a steep positive gradient, leveling out slowly after a peak. The probability density function of a log-normal distribution produces this kind of curve; log-normal and normal distributions are closely related. If the random variable Y is normal distributed, then $X = \exp(Y)$ is log-normal distributed. The expected value $E[X]$ and the variance $Var[X]$ are

$$E[X] = \exp\left(\mu + \frac{\sigma^2}{2}\right) \quad (41.1)$$

and

$$Var[X] = \exp(2(\mu + \sigma^2)) - \exp(2\mu + \sigma^2). \quad (41.2)$$

Conversely, if the expected value and variance is given, μ and σ of the associated normal distribution can be obtained as follows:

$$\sigma = \sqrt{\log\left(1 + \frac{\text{Var}[X]}{(E[X])^2}\right)} \quad (41.3)$$

and

$$\mu = \log(E[X] - \frac{1}{2}\sigma^2). \quad (41.4)$$

If users wish to generate a population with departure times following a log-normal distribution, it is hard to see how σ and μ will determine the outcome. It is much more convenient to consider expected value and variance. Given Equation (41.3) and Equation (41.4), a conversion from expected value and variance to σ and μ is straightforward.

41.6.2 Evacuation Area

The ScenarioManager integrates modules for the evacuation area definition and distribution of the affected population. The so-called evacuation area selection module allows the user to define the evacuation area by drawing either a simple polygon or circle on a map. The application can make use of either a WMS-provider or a tile map provider (e.g., OSM) as background map renderer. Zooming and panning is restricted to the bounding box of the OSM network file provided in the scenario configuration. An illustration of the evacuation area selector is given in Figure 41.2. In addition to defining a new evacuation area, a pre-existing one can also be loaded into the ScenarioManager. The requirements for a pre-existing evacuation area file are:

- It has to be provided as a ESRI shape file.
- The evacuation area must be defined as a simple polygon or a multi-polygon containing one, and only one, simple polygon.
- The coordinate reference system for polygon in the ESRI shape file must be set correctly.

Due to the high likelihood of error, this approach is recommended for experienced users only.

Later in the process, the ScenarioManager takes the evacuation area to cut out an evacuation network. However, after cutting out the evacuation net, there is no particular node as a target for the route calculation, as evacuees have more than one safe place as a destination. Instead, in the underlying domain, every node outside the evacuation area is a possible destination for an evacuee seeking an escape route. Thus, the evacuation problem is, in general, a multi-destination problem. To resolve this, the standard approach (e.g., Ford and Fulkerson (1962); Lu et al. (2005)) is to extend the network in the following way: All exit links (i.e., links that originate inside the evacuation area and terminate outside the evacuation area) are connected, using virtual links with very high (essentially infinite) flow capacity and equal length, to a super-node; all evacuation routes are routed to the super-node. This way, the problem is reduced to a multi-source single-destination problem. And thus, finding the shortest path from any node inside the evacuation area to this super-node and, in consequence, to safety, can efficiently be solved. For technical reasons, a super-link is added to the super-node and the evacuees are routed to that link (see the image at the beginning of this chapter).

41.6.3 Evacuation Demand

The process of defining the population distribution is similar to that of the evacuation area, differing because population is distributed over circles drawn on the map. The user can draw

in MATSim agents depart on links, so the `ScenarioManager` calls the `getNearestLink()` method defined in `NetworkImpl`. Thus, agents will depart on links inside and possibly near the circular areas.

In the current version, it is impossible to use a predefined demand for the simulation. Extending the simulation package in this way would be straightforward, but is out of this work's scope.

41.6.4 Road Closures

In real situations, some evacuation roads might not be available for the evacuation, because:

- They might be impassable due to the event (often the case in flooding-related evacuations).
- The authorities might want to keep roads open only for action/help traffic.
- In some situations, like hurricane evacuations, lane direction on motorways might be reversed to increase flow capacity in one direction.
- The authorities have detailed evacuation plans in place, with pre-planned evacuation routes; road closures might be necessary to force evacuees onto certain routes.

The actual planning of road closures can be a complex undertaking; not all attributes can be integrated into a simple tool for rapid evacuation planning. Nevertheless, the `ScenarioManager` offers a tool to create time-dependent road closures. An illustration of the road closures editor is given in Figure 41.3(a).

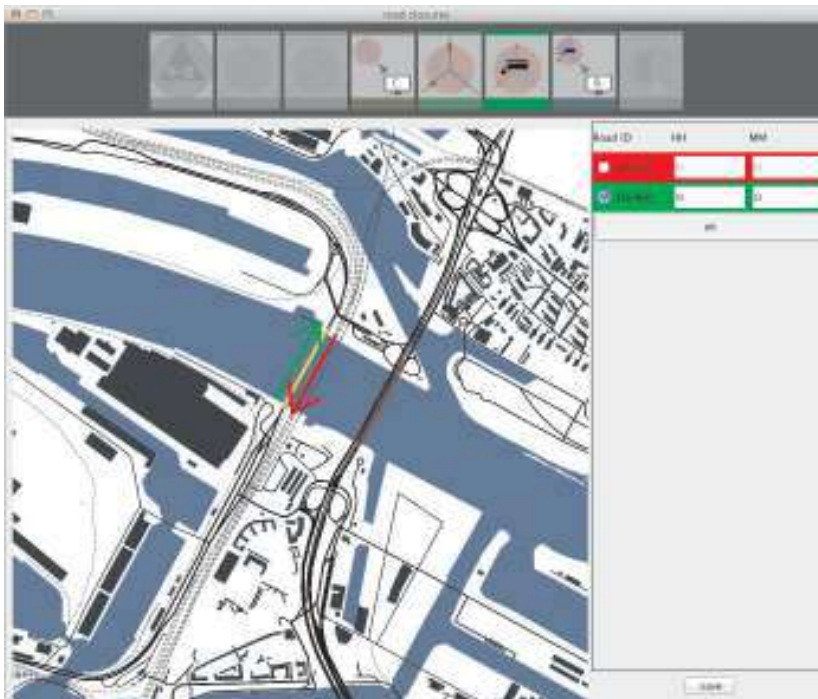
Road closures are stored as `NetworkChangeEvents` and handled as time-dependent network attributes in MATSim (Lämmel et al., 2010).

41.6.5 Bus Stop Editor

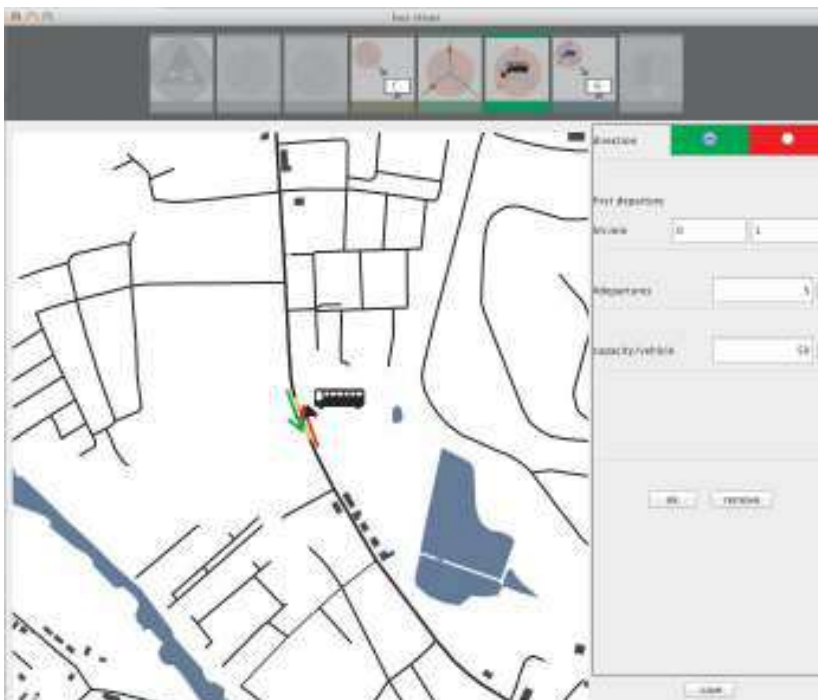
Usually, not everyone has access to a private car. In the event of an evacuation, those people often rely on public transport. In regions prone to natural disasters, local authorities normally have detailed evacuation plans in place, probably including evacuation by public transport. Consequently, it is important to have a tool available to help integrate public transport into the simulation scenario. The `ScenarioManager` offers this possibility by defining bus stops and bus schedules in the interactive GUI. Figure 41.3(b) gives an example of the bus stop editor. In addition to location, the user can define when the first bus will serve a bus stop, how many buses overall will serve this particular bus stop and these buses' capacity. The `ScenarioManager` transforms the inputs made into the GUI into a MATSim compatible transport schedule, enriching the scenario while using the same simulation model. Details about public transport simulations with MATSim are given in Chapter 16. A tutorial can be found on the MATSim webpage <http://matsim.org/docs/tutorials/transit>.

Limitations of the public transport evacuation approach in this project are:

- Each bus serves one and only one bus stop, perhaps a realistic assumption.
- Buses always take the shortest path from their designated bus stops to the safe area. As the shortest path is not necessarily the fastest, this approach might lead to avoidable delays. Some newer research investigates optimization of bus lines with respect to traffic demand and traffic conditions (Neumann, 2014). Implementing such optimization techniques in the evacuation context is a topic of future research.



(a) Road closures.



(b) Bus stop locations and schedules.

Figure 41.3: Top: Road closures can be edited by an integrated GIS application. For every link the direction and the time of closure can be defined. Bottom: Tool that let the user define bus stop locations and schedules.

41.6.6 *Running the Scenario*

The `ScenarioManager` runs the evacuation simulation in a way similar to other transport simulation studies with MATSim. At the beginning, an evacuation plan is assigned to each evacuee. An evacuation plan describes how the evacuee intends to reach the safe area. If the evacuee leaves by car or on foot, the plan is essentially comprised of a route (typically the shortest) from home to the safe area. For evacuees who depart by public transportation, the plan can be much more complex. All these evacuation plans will be executed in the mobility simulation; after this terminates, all plans are scored by travel time. The shorter a plan's travel time is, the higher is the score it receives. After this step, evacuees' plans are revised; some will receive new plans, while others continue with the current ones. This step is called re-planning. Mobility simulation, scoring, and re-planning are repeated in a loop for a predefined number of iterations; evacuees' individual performance improves over the iterations. In general transport studies, this approach emulates real-world travelers' behavior when they perform their daily commutes and try to find better travel alternatives. Evacuations, however, are singular events where such day-to-day re-planning would not occur. We argue here that the chosen iterative learning approach could be seen as the evacuees' anticipation of the conditions expected during an evacuation. People familiar with their surroundings would probably avoid roads that obviously constitute bottlenecks during an evacuation. Nevertheless, far more research is needed to definitively answer how people choose evacuation routes, or how many learning iterations are required to realistically reflect assumed anticipation skills adequately. As a rule of thumb, running 100 learning iterations are usually sufficient to achieve results constituting a lower evacuation times boundary.

41.6.7 *Analysis*

After the last iteration has finished, the `ScenarioManager` enables the analysis module. The analysis model evaluates the performed simulation run, using a number of different methods.

- The cumulative arrival curve tells the user the number of persons evacuated over time. From this curve, the user can, for example, learn at what time 50 % of the population has reached a safe destination.
- The GIS-based evacuation time analysis draws a grid over the evacuation area and computes, for every grid cell, average evacuation time. The evacuation times are indicated by different colors; the analysis modules run a quantiles-based clustering analysis for each cell. The size of cells can be varied by the user.
- The GIS-based clearance time analysis is performed in the same way as the evacuation time analysis. The clearance time of a cell is the time when the last evacuee leaves that cell. This evacuee is not necessarily the one who also started his/her evacuation inside the corresponding cell, but might also be one who crosses that cell somewhere during the evacuation.
- A similar quantiles-based clustering approach is used for the link utilization analysis. The link utilization analysis results help the user to identify the major evacuation routes.

The analyses can be run for every single iteration for which the MATSim Controller has dumped an events file (every 10th iteration by default). An overview of the analysis module is given in Figure 41.4

41.7 Conclusion

This chapter demonstrates how rapid evacuation planning can be performed with help of the evacuation contribution. The evacuation contribution provides an interactive GUI to perform this task. The only required external input is a network file extracted from OSM, thus a simple scenario

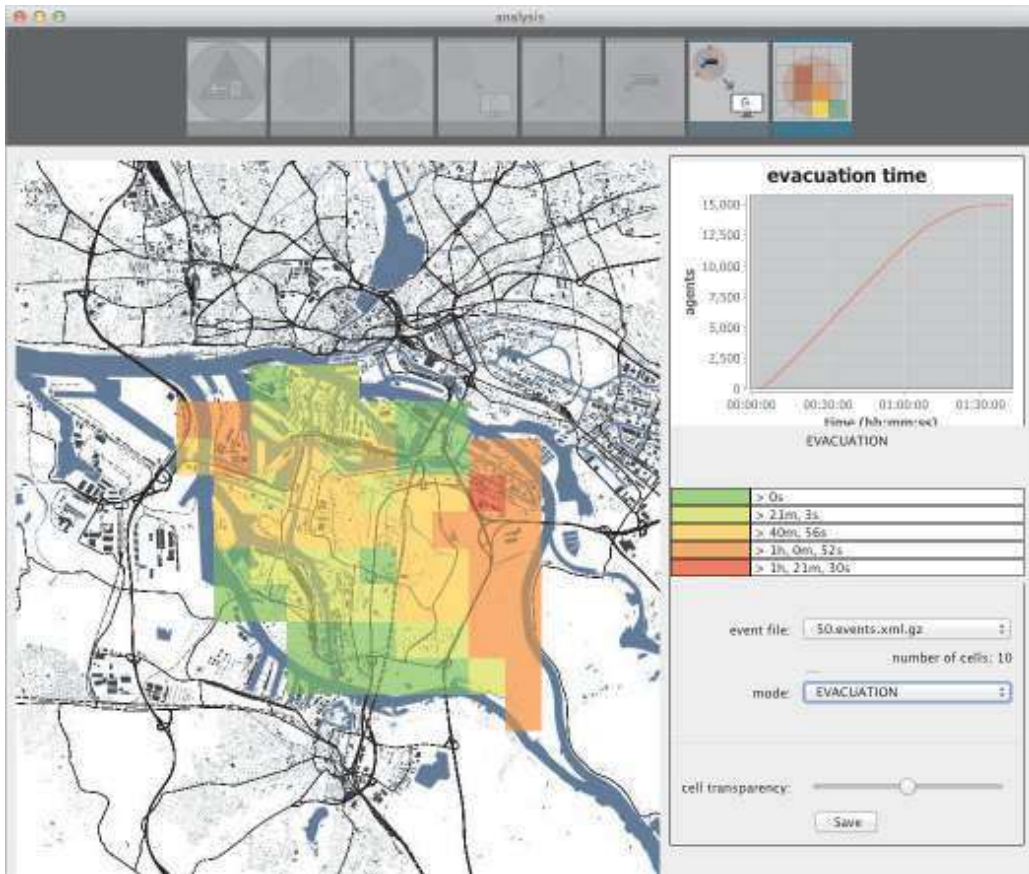


Figure 41.4: Screenshot of the analysis module showing GIS-based evacuation time analysis and the evacuation curve.

can be setup, simulated, and analyzed in less than an hour. Obviously, for an in-depth evacuation analysis of a certain area, a sort of expert knowledge is needed that a simple GUI can not supply. Still, for a rapid appraisal and for demonstration purposes, evacuation offers a powerful and easy-to-use tool. In the future, we plan to integrate a more advanced public transport planning tool based on Neumann (2014). Work is also ongoing to develop a more sophisticated pedestrian simulation model based on the theoretical framework given in Flötteröd and Lämmel (2015).

CHAPTER 42

MATSim4UrbanSim

Kai Nagel

42.1 Basic Information

Entry point to documentation:

<http://matsim.org/extensions> → matsim4urbansim

Invoking the module:

The module is invoked from a live UrbanSim implementation.

Selected publications:

Nicolai et al. (2011); Nicolai and Nagel (2014); Nicolai and Nagel (2015)

42.2 Summary

“MATSim4UrbanSim” is an adapter package for using MATSim as a travel model plug-in to UrbanSim, a well-known land use simulation (e.g., Waddell et al., 2003, see <http://www.urbansim.org>). UrbanSim has, for example, submodels for residential location choice, commercial location choice, or development and building construction, thus creating synthetic potential urban or regional development scenarios under various conditions and constraints. Traffic infrastructure plays a significant role in such developments; for example, very accessible areas are more attractive as residences and for commercial activities. Since accessibility is reduced by congestion, and congestion can only be realistically modeled through a sophisticated model of demand and supply interaction, UrbanSim does not have its own travel model, but delegates that task to external models, such as MATSim.

To use MATSim4UrbanSim, one first needs to have a running UrbanSim installation. From there, one can add MATSim to that installation; see the documentation mentioned above for more

How to cite this book chapter:

Nagel, K. 2016. MATSim4UrbanSim. In: Horni, A, Nagel, K and Axhausen, K W. (eds.) *The Multi-Agent Transport Simulation MATSim*, Pp. 283–284. London: Ubiquity Press. DOI: <http://dx.doi.org/10.5334/baw.42>. License: CC-BY 4.0

information. Basic MATSim parameters are configured from the UrbanSim configuration file by adding an appropriate section; again, see the documentation mentioned above for more information. It is possible to add a standard MATSim config file allowing use of the extended MATSim features, including those added after the adapter package was designed.

The module was applied by Cabrita et al. (2015) and by Zöllig Renner (2014).

CHAPTER 43

Discontinued Modules

Kai Nagel and Andreas Horni

This chapter lists modules that were important for several projects in the past, but which are no longer being developed.

43.1 DEQSim

DEQSim was used for project *Westumfahrung* (Balmer et al., 2009a). It was a queue-based, event-based parallel simulation written in C++ (Charypar et al., 2007b; Charypar, 2008). This simulation included handling of reduced capacities due to traffic lights in an aggregate manner (Charypar, 2008, p.139 ff). It also supported modeling of gap back propagation at junctions (Charypar, 2008, p.98 ff).

Events were written to file by DEQSim and subsequently read by MATSim. This represented a major framework performance bottleneck. DEQSim was therefore replaced by a Java version, the JDEQSim (see Section 4.3.2).

43.2 Planomat

Chapter 45 explains how MATSim can be extended. One long-standing extension point is the PlanStrategy extension point (Section 45.2.9). It allows the addition of “innovative” strategy modules (see Section 4.5), above and beyond those available by default.

One such replanning model was Planomat (Meister et al., 2006; Meister, 2011). It replaced the randomizing modules for (departure) time innovation (Section 4.5.1.1) and for mode innovation (Section 4.5.1.3), with a module that computed a joint best reply for both choice dimensions internally, using a Genetic Algorithm. Thus, it evaluated not just one random alternative per iteration, as standard MATSim would do, but multiple alternatives within one single iteration, to obtain an (at least locally) optimal solution. Planomat was successfully applied in the project

How to cite this book chapter:

Nagel, K and Horni, A. 2016. Discontinued Modules. In: Horni, A, Nagel, K and Axhausen, K W. (eds.) *The Multi-Agent Transport Simulation MATSim*, Pp. 285–286. London: Ubiquity Press. DOI: <http://dx.doi.org/10.5334/baw.43>. License: CC-BY 4.0

“KTI Frequencies” for time and mode innovation for sub-tours (Balmer et al., 2010, p.10). Unfortunately, there were three interacting problem complexes with Planomat:

- Any strategy module generating best reply plans must be able to compare plans and select a better one, at least along the considered choice dimensions. This is typically achieved by giving such a module an objective function which needs to be optimized. For example, a fastest path router minimizes the travel time; a generalized cost router minimizes the generalized travel cost.

All best reply modules here face the challenge that they cannot run the full mobsim (= network loading = synthetic reality) every time they need such information. As a result, all best reply modules are forced to build some internal synthetic reality model.

Planomat did this by running its plans through a simplified mobsim of its own. This mobsim was a re-implementation of the most important aspects for the core mobsim. Unfortunately, however, this meant that Planomat would not automatically pick up any change or addition to the core mobsim. Consequently, Planomat’s idea of a good plan often diverged from MATSim’s, especially when MATSim extensions were used. In other words, any addition to the MATSim system: e.g., tolls, or opening/closing time restrictions, or differentiating link travel times by turning directions, would have to be mirrored inside Planomat.

- Planomat always tended to return the same solution: understandable from a best-reply module, but it becomes a problem when what the module thinks is a best reply starts to differ from what the MATSim core thinks.

While an innovative strategy that deliberately generates diversity can be useful even when not fully consistent with the MATSim core (Nagel et al., 2014), this cannot function with a non-diverse innovative strategy, since it then insists on returning only suboptimal plans.

- In addition, Planomat used the MATSim core router in a way that hindered further software development of the core router. Essentially, Planomat used MATSim classes and methods that were not designed for re-use, but just happened to be public.

It was thus an obstacle for a major MATSim core router re-design, undertaken by T. Dubernet (see Section 45.2.7).

The combination of these three issues meant that Planomat was eventually discarded: Moving it to the new router infrastructure would have entailed a major piece of one-time work. After that, maintaining Planomat’s best-reply capability would have been a permanent work-intensive obligation. It was thus decided instead to invest our scarce resources in the design of a better core, allowing extensions to survive without much manual intervention. Although this will always be work in progress, Chapter 45 explains our substantial progress toward pluggable extensibility.

However, it must be noted that the improved software architecture does not resolve the general conceptual problem; best reply modules somehow need to follow core system development. Chapter 39 discusses a newer alternative that re-uses mobsim output for plan evaluation without having to run the full mobsim every time. An alternative approach, based on plan diversity, is investigated by Nagel et al. (2014). Additionally, Chapter 49 discusses aspects of diversity in plan set generation.

43.3 PlanomatX

PlanomatX was based on Planomat. It extended it by performing activity choice and adopting a Tabu Search approach (Feil, 2010). To cope with the curse of dimensionality (due to the added choice dimension), PlanomatX introduced schedule recycling, basically a warmstart concept. Because of problems when using the standard MATSim logarithmic utility function for activity choice, PlanomatX also derived an alternative utility function from Joh (2004). Rough estimates for its parameters based on an MNL exist, but turned out to be problematic, as shown in Section 97.4.3.

PlanomatX, derived from Planomat, suffered from the same maintenance problems and was eventually abandoned for the same reasons.

SUBPART THIRTEEN

Development Process & Own Modules

Organization: Development Process, Code Structure and Contributing to MATSim

Marcel Rieser, Andreas Horni and Kai Nagel

This chapter describes how new functionality enters MATSim. It describes the MATSim team and community, the different roles existing in the MATSim project, the development drivers and processes, and the tools used for integration. The goal is to provide an overview of the development process so that one quickly finds access to the MATSim community and is able to efficiently contribute to MATSim, based on one role or another.

44.1 MATSim's Team, Core Developers Group, and Community

The **MATSim team** currently consists of three research groups and a spin-off company:

- the VSP (VerkehrsSystemPlanung und Verkehrstelematik – The Transport Systems Planning and Transport Telematics group at TU Berlin) group at the ILS (Institut für Land- und Seeverkehr – Institute for Land and Sea Transport Systems), TU Berlin, led by Prof. Dr. Kai Nagel,
- the VPL (VerkehrsPLANung) group at the IVT, ETH Zürich, led by Prof. Dr. Kay W. Axhausen,
- the recently founded Mobility and Transportation Planning group at the FCL, based in Singapore and led by Prof. Dr. Kay W. Axhausen, and
- Senozon AG, based at Zürich with a subsidiary in Germany, founded by former PhD (Philosophiae Doctor – Doctor of Philosophy) and research students.

As is common in research, the university groups' composition changes frequently. Over the last decade more than 50 people, as listed earlier, contributed to MATSim.

How to cite this book chapter:

Rieser, M, Horni, A and Nagel, K. 2016. Organization: Development Process, Code Structure and Contributing to MATSim. In: Horni, A, Nagel, K and Axhausen, K W. (eds.) *The Multi-Agent Transport Simulation MATSim*, Pp. 289–296. London: Ubiquity Press. DOI: <http://dx.doi.org/10.5334/baw.44>. License: CC-BY 4.0

A small group of the MATSim team defines the **MATSim core developers group**, maintaining MATSim's core as defined below in Section 44.3.2.

In addition, there is a **MATSim community** composed of closely connected research groups in other cities, e.g., Stockholm, Pretoria, Poznan, and Jülich, as well as more loosely connected external users coming together, e.g., at the annual MATSim User Meeting (see Figure 44.1).

MATSim is open-source software under the GPLv2 (GNU General Public License version 2.0). You are also very welcome to contribute to the code base as described in Section 44.6. New contributors are mentored in the beginning to become familiar with the project and the coding conventions.

44.2 Roles in the MATSim Community

The MATSim community includes the following roles:

- The **MATSim user** uses the official releases or nightly builds and runs the MATSim core with the config file (Section 5.1.1). He or she does not write computer code. Part I of the book is dedicated to the MATSim user. On the web page, he or she finds relevant information in the *user's guide* section and in the user's mailing list users@matim.org.¹ There is also a list of questions and answers under <http://matim.org/faq>.

Users should also remember to consult the files `logfileWarningsErrors.log` and `output_config.xml.gz`, as also explained in Section 2.3. The former file is an extract from `logfile.log`, but only contains the warnings and errors. The latter is a complete dump of the currently available configuration options, including comments to most options.

- The **MATSim power user** is a MATSim user with knowledge on how to use the additional modules presented in the book's Part II. He or she does *not* program but knows how to use MATSim scripts-in-Java prepared by others or her/himself, as shown in Section 5.1.1. Parts I and II of the book are helpful to the MATSim power user. Information about extensions can be found under <http://matim.org/extensions>. Most extensions come with an example script-in-Java. Again, questions and answers are under <http://matim.org/faq>.
- The **MATSim developer** extends MATSim by programming against the MATSim API (Section 5.1.1). He or she also finds his or her information in Part II of the book, in particular, in Chapter 45, on the web page in the Developer's Guide, and in the mailing list developers@matim.org.
- There are relatively few **MATSim core developers** in the MATSim team. These persons make necessary modifications of the core (as defined in Section 44.3.2), usually after having discussed them in the issue tracker (<http://matim.org/issuetracker>), in the MATSim committee, or at a developer meeting (see below).

44.3 Code Base

The various pieces of MATSim are delineated by Apache Maven projects and sub-projects. The Apache Maven layout corresponds to the layout of the Git repository.² Note that the Java package structure does *not* directly correspond to the Apache Maven/Git layout.

¹ During the writing of this book, the information that had so far been contained in the User's Guide was moved to this book. Therefore, the User's Guide section on the web page is currently essentially empty, and may be removed.

² MATSim is currently at GitHub under <https://github.com/matim-org/matim>. The exact path name may change in the future, e.g., because of changes at GitHub.



Figure 44.1: MATSim events and community.

Source: ©Dr. Marcel Rieser, Senozon AG

44.3.1 Main Distribution

The “MATSim main distribution” corresponds to the “matsim” part of the Git repository. It is the part of the code that the MATSim team feels primarily responsible for. At the time of writing, the MATSim main distribution contains following packages:

- `org.matsim.analysis.*`, containing certain analysis packages that are added by default to every MATSim run.
- `org.matsim.api.*`, see Section 44.3.2.
- `org.matsim.core.*`, see Section 44.3.2.
- `org.matsim.counts.*`, see Section 6.3.
- `org.matsim.facilities.*`, see Section 6.4.
- `org.matsim.households.*`, see Section 6.5.
- `org.matsim.jaxb.*`, containing automatically or semi-automatically generated adapter classes to read XML files using JAXB (Java Architecture for XML Binding).
- `org.matsim.lanes.*`, see Chapter 12.
- `org.matsim.matrices.*`, containing (somewhat ancient) helper classes to deal with matrices, in particular, origin-destination-matrices.
- `org.matsim.population.*`, mostly containing a collection of algorithms that go through the population and modify persons or plans.
- `org.matsim.pt.*`, see Chapter 16.
- `org.matsim.run`, see Section 44.3.2.
- `org.matsim.utils.*` containing various utilities such as the much-used `ObjectAttributes` (see Section 45.2.2).
- `org.matsim.vehicles.*`, see Section 6.6.
- `org.matsim.vis.*`, containing helper classes to write MATSim information, in particular from the `mobsim`, to file. This has to a large extent been superseded by the `Via` visualization package (see Chapter 33).
- `org.matsim.visum.*`, containing code to input data from VISUM.
- `org.matsim.withinday.*`, see Chapter 30.
- `tutorial.*`, containing example code of how to use MATSim, referenced throughout this book.

44.3.2 Core

The core is part of the main distribution (see the previous Section 44.3.1) and contains material that is considered basic and indispensable, and resides in the packages

- `org.matsim.api.*`
- `org.matsim.core.*`
- `org.matsim.run.*`

The MATSim core is maintained by the MATSim Core Developers Group.

44.3.3 Contributions

The idea of the contributions part of the repository is to host community contributions. Historically, most contributors are from the MATSim team, but this is not a requirement.³ The

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

code is maintained by the corresponding contributor. Code in this section of the repository is considered more stable than code in playgrounds. The Java packages often have the root `org.matsim.contrib.*`, but this is not mandatory.

At the time of writing, there are the following contributions (= extensions which are in the “contrib” part of the repository), listed in alphabetical order:

- `accessibility`, presented in Chapter 35.
- `analysis`, presented in Chapter 38.
- `cadytsIntegration`, presented in Chapter 32.
- `common` is not a true contrib, i.e., it does not provide additional functionality by itself. Instead, it is a place where code used by several contribs, which has not yet made it into the main distribution is located. It also contains some long-running integration tests that are run at each build (i.e., more often than those contained in the integration contrib described below).
- `dvrp`, presented in Chapter 23.
- `emissions`, presented in Chapter 36.
- `freight`, presented in Chapter 24.
- `freightChainsFromTravelDiaries`, presented in Chapter 26.
- `grips`, presented in Chapter 41.
- `gtfs2matsimtransitschedule`, presented in Chapter 18.
- `integration` is not a true contrib, i.e., it does not provide additional functionality. Instead, it is a place where integration tests that should run daily or weekly (instead of as often as possible) can be committed.
- `locationchoice`, presented in Chapter 27.
- `matrixbasedptrouter`, presented in Chapter 20.
- `matsim4urbansim`, presented in Chapter 42.
- `minibus`, presented in Chapter 17.
- `multimodal`, presented in Chapter 21.
- `networkEditor`, presented in Chapter 10.
- `otfvis`, presented in Chapter 34.
- `parking`, presented in Chapter 13.
- `roadpricing`, presented in Chapter 15.
- `socnetgen`, presented in Chapter 29.
- `socnetsim`, presented in Chapter 28.
- `transEnergySim`, presented in Chapter 14.
- `wagonSim`, presented in Chapter 25.

44.3.4 *Playgrounds*

Another element of the MATSim repository is the “playgrounds”. These are meant as a service to programmers. They have grown historically from the fact that MATSim’s object classes and in consequence the interfaces between them have evolved and grown over time, and thus a stable API was not available. Regular code-wide refactorings, along the lines discussed, e.g., by Fowler (2004), were thus the norm for many years.

At this point, the extension points described in Chapter 45 should be somewhat stable and development against them should be possible without major changes from release to release. Anybody who needs tighter integration with the project should still apply for a playground.

44.3.5 Contributions and Extensions

Congruent with the structure of this book, the MATSim code structure contains a core which allows to run basic MATSim using the config file, a population and a network. Packages going beyond this basic functionality are extensions, where three different kind of extensions exist:

- extensions in the main distribution,⁴
- extensions contributed by the MATSim community known as contributions, and
- any code written anywhere published or unpublished extending the MATSim core.

Extensions are listed at <http://matsim.org/extensions>.

44.3.6 Releases, Nightly Builds and Code HEAD

Releases, nightly builds and the code head can be obtained from <http://matsim.org/downloads>.

MATSim releases are published approximately annually. Usually, MATSim users and MATSim power users as defined above in Section 44.2 work with releases.

MATSim uses continuous integration and, thus, nightly builds are available without stability guarantee under <http://matsim.org/downloads/nightly>. MATSim API developers that depend on a very recent feature might use Nightly builds.

Both Apache Maven releases and Apache Maven snapshots are available, see <http://matsim.org/downloads> for details.

MATSim API developers or core developers often work on the code's HEAD version that can be checked out from GitHub.

Nightly builds and maven snapshots are only generated when the code compiles and passes the regression tests. They are, in consequence, somewhat “safer” than the direct download from the HEAD.

44.4 Drivers, Organization and Tools of Development

Important drivers of the MATSim development are the projects and dissertations of the MATSim team. New features are developed as an answer to requirements of these dissertations and projects, where projects range from purely scientific ones—often sponsored by SNF (Schweizerischer Nationalfonds) or DFG (Deutsche Forschungsgemeinschaft)—via projects for governmental entities and projects where science and industry contribute equally—e.g., CTI (Commission for Technology and Innovation) projects—to purely commercial projects, which are managed by Senozon AG in the majority of cases. A significant number of innovations are also introduced by the collaboration with external researchers.

Systematic code integration is mainly performed by the Berlin group and by Senozon AG. This includes continuous code review and integration upon request of the community, but also comprehensive code refactorings to clean up code and to improve modularity. Refactorings are discussed and documented in the MATSim issue tracker (<http://matsim.org/issuetracker>).

The development process is supported by a MATSim standing committee discussing software and sometimes conceptual issues on a regular basis (<http://matsim.org/committee>). Another element that brings in innovation as well as organization are the annual meetings. Right from the beginning, there have been a MATSim developer meetings focused on coding issues. Later, a user meeting offering insights into current work by the community has been added, sometimes

⁴ At the time of writing it is unclear if these extensions might one day become contributions, shrinking the MATSim main distribution to its core.

combined with a tutorial. Finally, a conceptual meeting is now held every year, concentrating on issues that go beyond pure software engineering. The developer meeting and the conceptual meeting together establish the road map that guides development for the remainder of the year.

MATSim development makes use of a large number of tools, hopefully leading to better software quality. Historically, many of those tools ran from automated scripts and were made available at <http://matsim.org/developer>. Nowadays, most of them are automatically available from the build server (see <http://matsim.org/buildserver>) and/or from the repository (<https://github.com/matsim-org/matsim>), so that many of them are scheduled for removal from <http://matsim.org/developer>. Some of these tools are: a change log; an issue tracker; the javadoc documentation; static code analyses performed by *FindBugs* and *PMD*; test code coverage analysis; copy paste analysis; code metrics; Apache Maven dependencies; and information about the nightly test results. These nightly test results are generated by the MATSim build server based on the Jenkins software.

Furthermore, there is a MATSim benchmark at <http://matsim.org/files/benchmark/benchmark.zip>. For results see <http://matsim.org/benchmark>.

Most MATSim developers use Eclipse as an IDE. The MATSim documentation is tailored to this IDE. Team development is currently based on Git as revision control system. External library dependencies are managed by Apache Maven.

44.5 Documentation, Dissemination and Support

The main documentation is now this book. Additional information, including tutorials, can be found under <http://matsim.org/docs>. Code documentation in form of javadoc can be found under <http://matsim.org/javadoc>.

For fast application of MATSim, some small-scale example scenarios are provided in the code base (folder: `examples`), where recently an extended version of the well-known benchmark scenario for the City of Sioux Falls has been added (Chakirov and Fourie, 2014) (Chapter 59). Additional example datasets, including Berlin datasets, can be obtained via <http://matsim.org/datasets>.

Further information is disseminated at the afore-described annual user meetings and MATSim mailing lists, see <http://matsim.org/maillinglists>. Support is provided by the MATSim team via these mailing lists and via <http://matsim.org/faq>, both on a best effort basis. Many components of MATSim are documented by the numerous papers published in international journals and presented at worldwide conferences. Information about such publications can, e.g., be obtained from <http://matsim.org/publications> and from this book.

44.6 Your Contribution to MATSim

The technical details, i.e., the MATSim extension points, on where to hook with MATSim are detailed in Chapter 45. Here, the different ways of contributing to MATSim according to the roles presented in Section 44.2 are introduced.

As a MATSim user, power user, or API developer, you are warmly welcome to make an important impact by reporting your achievements, needs and problems with, or bugs of, the software via the users mailing list, the issue tracker, the FAQ, or at the annual MATSim user meeting.

If you would like to directly contribute to the code base of MATSim, you are welcome to become part of the contributions repository.

If you are the type of person that likes to change the core system, you can, although it is a long way, become a member of the MATSim core developers group. Core developers are usually picked from the MATSim team. Prerequisites are a strong computer scientist background, several years of experience with MATSim and a deep understanding of large software projects.

CHAPTER 45

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

Michael Zilske

Notes

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

For programming against the MATSim API, we recommend <https://github.com/matsim-org/matsim-example-project> as a starting point; in particular, this should clarify how MATSim can be used as an Apache Maven plug-in.

45.1 Introduction

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

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

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

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

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

How to cite this book chapter:

Zilske, M. 2016. How to Write Your Own Extensions and Possibly Contribute Them to MATSim. In: Horni, A, Nagel, K and Axhausen, K W. (eds.) *The Multi-Agent Transport Simulation MATSim*, Pp. 297–304. London: Ubiquity Press. DOI: <http://dx.doi.org/10.5334/baw.45>. License: CC-BY 4.0

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

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

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

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

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

As a result, it is possible to add additional executable code into the execution flow of the QSim by MobsimListeners in a similar way as this is possible with the ControllerListeners mentioned above. The router, in contrast, is most importantly configured by replacing the definition of the generalized travel cost.

45.2 Extension Points

This section describes what could be called the SPI (Service Provider Interface) of MATSim. Historically, the main entry-point for writing a MATSim extension has been to literally extend (in the Java sense, i.e. to inherit from) the Controller class. Essentially, one would override the methods calling the mobsim, the scoring, and/or the replanning, as explained in Section 45.1. This is now discouraged. While this pattern worked when each member of the team was working on extending the MATSim core by a different aspect, it fails when it comes to integrating those aspects to a single product: There is nothing one can do with a PublicTransportController, an EmissionsController, a RoadPricingController and an OTFVisController, if one wants to combine them to visualize the emissions of buses on toll roads. Also see Section 46.2.1.5.

45.2.1 Config Group

The configuration of a MATSim run is a grouped list of key-value pairs, stored in XML format in the config file (see Section 45.2.1).

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

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

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

See <http://matsim.org/javadoc> → main distribution → RunReflectiveConfigGroup for an example.

45.2.2 *ObjectAttributes and Customizable*

MATSim operates on data types such as links, nodes, persons, or vehicles. Many of these data types have attributes, such as free speed (for links) or coordinates (for nodes). Rather often, one would like additional information for certain data types, such as “slope” for links, or “age” for persons. In order to not modify the data types every time this becomes necessary, but still allow experimentation, a helper container called `ObjectAttributes` is available. It essentially attaches arbitrary additional information to objects *that have an ID*, by a syntax of type

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

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

Importantly, the package provides readers and writers for such attributes. It is thus possible for additional code to, say, generate additional attributes by preprocessing, write them to file, and read them back for every run. That approach is used, for example, by the Gauteng scenario (Chapter 69) to pre-allocate e-tag ownership to persons.

Note that there is currently no simple way to similarly attach information to data types that do not have an ID. This is, for example, the case with plans, activities, or legs, which are contained inside a data type with an ID (the person data type), but which do not possess an ID of their own and are therefore not addressable by `ObjectAttributes`. Some of these non-identifiable data types implement the Java interface `Customizable`, to which additional material can be attached by a syntax of type

```
plan.getCustomAttributes.put( "myAttribName", myAttribValue ) ;
```

For additional information, see the `Customizable` interface under <http://matsim.org/javadoc> → main distribution. Note that information contained in `Customizable` is not considered standard information by MATSim. It is not written to file when writing the corresponding container, it is in consequence not read from file, and it is undefined if it is copied when copying the data object (e.g., when cloning plans for the evolutionary algorithm). This is the status quo; the MATSim team is thinking about better solutions.

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

45.2.3 *Scenario Element*

The object-oriented, in-memory database which holds the Person objects with their plan memories is accessible via the `Population` interface. The `Network` interface gives access to the traffic network graph, consisting of links and nodes. There is a `TransitSchedule` interface which represents the public transit schedule. Your own modeling tasks may need an additional data container like these. We call them scenario elements. The freight carrier population of the freight extension described in Section 24.2 is a typical example.

Scenario is the interface which ties all scenario elements together. You can add your own named scenario element to the Scenario at startup, for example in a `StartupListener`. All standard scenario elements are populated from XML files at startup, but your own scenario elements could just as well be interfaces to an external relational database.

See <http://matsim.org/javadoc> → main distribution → `RunScenarioElementExample` for an example. Note, however, that in the meantime, the injection framework may have become a better alternative.

45.2.4 *ControllerListener: Handling Controller Events*

Controller remains the main user-facing class of MATSim, but please do not subclass it. Rather, use its setter methods to plug in your own code.¹

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

The following ControllerListeners are currently available: StartupListener, IterationStartsListener, BeforeMobsimListener, AfterMobsimListener, ScoringListener, IterationEndsListener, ReplanningListener, and ShutdownListener. An up-to-date list can be obtained from <http://matsim.org/javadoc> → main distribution → ControllerListener interface.

A sample listener might look as follows.

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

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

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

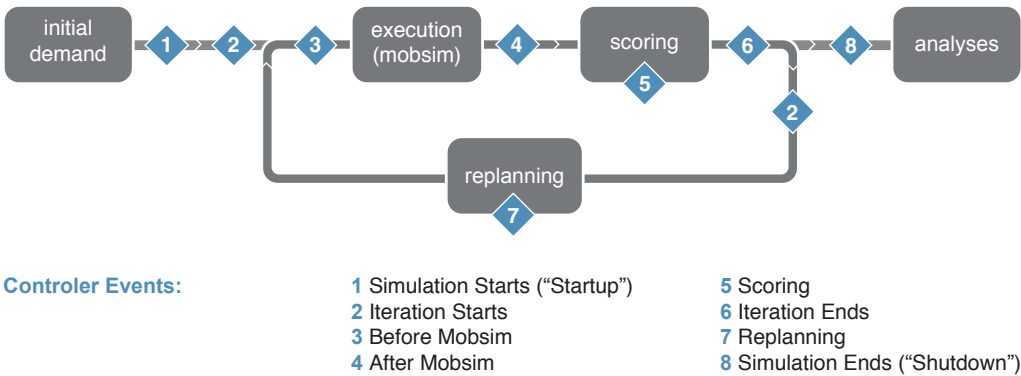


Figure 45.1: Controller events.

¹ Again, in the meantime, the injection framework may have become a better alternative altogether. The general structure, however, remains the same.

45.2.5 Events

The mobsim moves the agents around in the virtual world according to their plans and within the bounds of the simulated reality. It documents their moves by producing a stream of events. Events are small pieces of information describing the action of an object at a specific time. Examples of such events are (also see Figure 2.2):

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

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

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

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

Handling Events MATSim extensions can watch the mobsim by interpreting the stream of events. This is done by implementing the `EventHandler` interface and registering the implementation with the framework. The lifecycle of an `EventHandler` can be chosen by the developer. Normally, an `EventHandler` lives as long as the simulation run. It is notified before the beginning of each new iteration so that its state can be reset to listen to a new iteration. This pattern can be used to collect information over all iterations. But if the purpose of an `EventHandler` is to make a calculation based on one single iteration, it may be more natural to create a new `EventHandler` instance for each iteration, query it for its result and discard it after the iteration finishes. This can be done in a `ControllerListener`.

See <http://matsim.org/javadoc> → main distribution → `EventHandler` for pointers to coding examples.

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

LinkEnterEvent for the same vehicle gives a model for a vehicle crossing a node. At this point, your code would look at the road network model to determine if this was a left-turn, and if so, produce a LeftTurnEvent.

See <http://matsim.org/javadoc> → tutorial → RunCustomScoringExample for an example.

45.2.6 Mobsim Listener

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

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

Example See <http://matsim.org/javadoc> → main distribution → MobsimListener for more details and pointers to examples.

45.2.7 TripRouter

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

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

- compute a route for a given mode and O-D pair, for a Person with a specific departure time,
- identify the *main mode* of a trip. For instance, a trip composed of several *walk* and *pt* legs should be identified as a public transport trip.
- Identify which activities are *stage activities*, and, by extension, identify the trips in the plan: A trip is the longest sequence of consecutive Legs and *stage activities*

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

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

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

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

45.2.8 *Mobsim*

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

Example for alternative mobsim in Java Please check the documentation of Mobsim (see <http://matsim.org/javadoc> → main distribution) for more details and pointers to examples.

Alternative mobsim in another programming language Your implementation need not be written in Java. The framework includes a helper class to call an arbitrary executable which is then expected to write its event stream into a file. This pattern has been used successfully many times, see, e.g., Section 43.1, or the CUDA implementation of the mobsim by Strippgen (2009). Note that we have found consistently that an external mobsim does *not* help with computing speed: Whatever is gained in the mobsim itself is more than lost again by the necessary data transfer between MATSim and the external mobsim. As of now, we cannot yet say if newer data exchange techniques, such as Google Protocol Buffers, may change the situation. Until then, the external interface should rather be seen as the option to inject a different, possibly more realistic, mobsim into MATSim.

45.2.9 *PlanStrategy*

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

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

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

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

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

The four most commonly used strategies shipped with MATSim are:

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

Routes are computed based on the traffic conditions of the previous iteration, which are measured by means of an `EventHandler`. Using the same pattern, your own `PlanStrategy` can use

any data which can be computed from the mobility simulation. The source code of the standard `PlanStrategy` implementations can be used as a starting point for implementing custom behavior.

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

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

45.2.10 Scoring

The parameters of the default MATSim scoring function (Chapter 3) are configurable. The code, which maps a stream of mobsim events to a score for each agent is placed behind a factory interface and replaceable. However, replacing it means replacing the entire utility formulation. There is currently no mechanism for composing a utility formulation from contributions by different modules. For instance, a module which simulates weather conditions would probably calculate penalties for pedestrians walking in heavy rain, and the Cadyts (Chapter 32) calibration scheme already uses utility offsets in its formulation. A modeler who wishes to compose a scoring function from the Charypar-Nagel utility, the rain penalty and the calibration offset needs to do this manually, in code, accessing the code of all three modules contributing to the score and (for instance) summing up their contributions. As of the writing of this chapter, this makes scoring in a way the least modular part of MATSim: It has to be re-defined, in code, for every combination of modules which contribute to the utility.

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

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

PART III

Understanding MATSim



CHAPTER 46

Some History of MATSim

Kai Nagel and Kay W. Axhausen

46.1 Scientific Sources of MATSim

As sketched earlier (Section 1.1), MATSim derives from the following research streams:

Microscopic Modeling of Traffic Microscopic modeling was a basis for traffic flow theory from the start (e.g., Herman et al., 1959; Seddon, 1972; Wiedemann, 1974), but the work was limited to individual links, or small sequences of links and could thus not address equilibrium, as aggregate assignment models could from the 1970's onward (see Sheffi, 1985; Ortúzar and Willumsen, 2011). The expansion to whole and large networks came with the increasingly powerful computers in the 1980's, as well as fast and sufficiently accurate flow models (e.g., Schwerdtfeger, 1984; Nagel and Schreckenberg, 1992; Daganzo, 1994; Gawron, 1998).

Computational Physics For MATSim, this development was aided by insights from computational physics, which often adopts simple and very fast models of physical processes and has performed simulations with 10^8 , and more, particles since the 1980's (for a contemporary review see Beazley et al., 1995). It was thus clear from the beginning that urban or regional systems with 10^7 or 10^8 persons or vehicles could be simulated microscopically; the research then focused on where necessary compromises would have to be made.

Microscopic Behavioral Modeling of Demand/Agent-Based Modeling According to Russel and Norvig (2010, p. 53), an agent is “anything that can be viewed as perceiving its environment through sensors and acting upon that environment through actuators”. In that sense, both the models of Seddon (1972) and of Wiedemann (1974) can be classified as agent-based; this holds even for the simple cellular automata models of Nagel and Schreckenberg (1992), since here driver-vehicle units perceive the distance to the vehicle ahead and act by adjusting their velocity.

Agent-based behavior can also be found at the demand modeling level, where aggregate models, such as the gravity model (Wilson, 1971), can be replaced by person-centric formulations.

How to cite this book chapter:

Nagel, K and Axhausen, K W. 2016. Some History of MATSim. In: Horni, A, Nagel, K and Axhausen, K W. (eds.) *The Multi-Agent Transport Simulation MATSim*, Pp. 307–314. London: Ubiquity Press. DOI: <http://dx.doi.org/10.5334/baw.46>. License: CC-BY 4.0

In that sense, agent-based modeling of travel demand had been developed in Germany since the 1970's (see the references in Axhausen and Herz, 1989), as well as in English-speaking countries, as described in Jones et al. (1983)'s seminal book. While anglophone authors focused on sample enumeration methods to estimate total demand with their activity-based demand models (see Bradley and Bowman (2006) for North American, mostly discrete choice, model-based, developments and Arentze and Timmermans (2000) for an alternative Dutch approach), the simpler German approach was linked to an integral mesoscopic traffic flow simulation in Axhausen (1989), but not used for equilibrium search. It already had, however, a simple description of daily schedule total utility.

Complex Adaptive Systems/Co-Evolutionary Algorithms Nash-equilibrium-like approaches had been developed in transport assignment since the formative Wardrop (1952) paper. These aggregate, flow-based approaches were expanded to account for user perception errors and the social optimum (see Daganzo and Sheffi, 1977). In the late 1990's, transport science addressed the process of learning within the context and new possibilities of "intelligent transport systems", using various smoothing techniques to integrate data from iteration to iteration, reflecting the field tradition. Examples include Chang and Mahmassani (1989); Kaufman et al. (1991); Hatcher and Mahmassani (1992); Smith et al. (1995); Axhausen et al. (1995); Nagel (1995, 1996); Gawron (1998); Mahmassani and Liu (1999); Polak and Oladeine (2002); Arentze and Timmermans (2004). These approaches translated Nash equilibrium logic into co-evolutionary search schemes, which efficiently identified the optima of each agent's daily schedule.

46.2 Stages of Development

46.2.1 Kai Nagel's Perspective

46.2.1.1 *Fast Microscopic Modeling of Traffic Flow (University of Cologne/Los Alamos National Laboratory)*

Kai Nagel originally wanted to do his PhD (Philosophiae Doctor – Doctor of Philosophy) in meteorology. When funding did not come through, he began exploring alternatives and applied for a position in insurance modeling with Prof. A. Bachem at the University of Cologne. Instead, he was offered a position in operations research, solving problems like dynamic vehicle routing with time windows.

Having some background in computational statistical physics, he soon became skeptical whether it made sense optimizing up to the last second of a time window, while simultaneously facing a highly stochastic transport system. Using his training, he embarked on building a microscopic model of the transport system, in particular single-lane (Nagel and Schreckenberg, 1992; Nagel, 1999) road traffic on long links, as well as combining such links to large-scale network-based simulations, where each vehicle follows its own individual route (Nagel, 1996), including adaptive dynamics, being influenced most heavily by Arthur (1994). That paper already (Nagel, 1996) describes what is still the main MATSim architecture, where agents have many different plans, keep trying them out and eventually settle on the best option. In contrast to the current approach, in that paper, all plans were pre-computed; i.e., there was no innovation during iterations. This was possible because the network was much coarser than what we use today, making pre-computing route plans with enough diversity easy.

46.2.1.2 *TRANSIMS (Los Alamos National Laboratory/Santa Fe Institute)*

Some of the above PhD work was done during Kai's tenure as a Graduate Research Assistant at LANL (Los Alamos National Laboratory). After his PhD, he moved to LANL, where he worked

with the TRANSIMS (see, e.g., Smith et al., 1995) team, under the leadership of Chris Barrett. The TRANSIMS project used some of the design described above, most notably the cellular automata approach to road traffic modeling, which was thus extended to multi-lane traffic (Nagel et al., 1998), to intersections (Nagel et al., 1997) and to massive parallel computing (Nagel and Rickert, 2001).

In terms of software design, TRANSIMS was a collection of stand-alone modules, coupled by a script. For example, the population synthesizer would generate a population file, the activity generator would take the population file as input and generate an activities file as output, etc. Iterations were done by running the traffic microsimulation (called *mobsim* in MATSim) based on plans and outputting average link travel times and then running the router based on link travel times and outputting plans.

46.2.1.3 *MATSim in C++ (ETH Zürich Computer Science)*

Kai Nagel moved to ETH Zürich Computer Science in 1999. It was difficult there to continue with TRANSIMS, partly because TRANSIMS was not under an open-source license at that time and also because TRANSIMS fell under U.S. technology export restrictions for some time. As a result, MATSim was started.

MATSim was different from TRANSIMS from the beginning in two important ways: (1) it tried to be more lightweight, i.e., running much faster, specifically by using the queue model (Gawron, 1998), rather than the cellular automata model for network loading and (2) other than TRANSIMS, agent properties such as demographic data, activity patterns or routes were no longer distributed across multiple files, but contained in one hierarchical XML file.

Another difference later appeared, which went back to the Nagel (1996) approach, but this time really followed Arthur (1994) by giving each individual agent its own memory (Raney and Nagel, 2006). After experimentation with relational databases such as MySQL (MySQL, accessed 2014) or Oracle (ORACLE [www page](http://www.oracle.com), accessed 2005), it was eventually decided to implement MATSim as an object-oriented database in memory, i.e., by first reading in all XML files, modifying the data in computer memory RAM during a run lifetime and writing the data back from memory to XML files at the end of the run. The decision was based on the observation that the MATSim data model was described much better by XML files and that conversion to the relational format was impractical, prone to errors, and too slow if not kept in memory during iterations.

46.2.1.4 *MATSim in Java (TU Berlin Transport Engineering)*

Michael Balmer wrote his dissertation at ETH (see below) about demand modeling for MATSim, i.e., about the upstream process that leads to initial plans (Balmer, 2007). That project, different from the main MATSim code at that time, was written in Java. Along with the assessment that Java would be the better language than C++ to continue development, it was decided to use Michael Balmer's code as starting point for a Java version. Arguments for Java included:

- Java is more restrictive. For example, in Java, objects are always passed by reference,¹ while in C++, one has the choice between passing a pointer, a reference, or a deep copy of the object. Since standards are difficult to enforce in academic environments, a more restrictive language seemed (and still seems) the better choice.
- Java runs well on many platforms. This allowed (and still allows) us to let people work on their favorite platforms, be it Linux, Microsoft Windows, or Mac.
- There is good non-commercial support for Java; for example, the Eclipse IDE and numerous powerful libraries.

¹ We abstract from the notion that Java “passes object references by value”.

- The Java compiler is easier to handle. For example, there is no extraction of header files and the Java compiler sorts out, by itself, the sequence in which modules need to be compiled.
- For our applications, Java was consistently *not* slower than C++. This assessment was based on several years of teaching a MATSim class at ETH Zürich, where computer science students implemented simple versions of MATSim in a programming language of their choice. Typically, while the fastest C++ code may have been 30 % faster than the fastest Java code, the slowest C++ code normally was a factor of 3 slower than the slowest Java code. In other words, while C++ gives more opportunities for optimization, it also gives more opportunities for very serious performance degradation. This assessment is corroborated somewhat in the literature (Prechelt, 1999), where, in one example, it is demonstrated that interpersonal differences within the same language are of the same magnitude as differences between languages.

In addition, it seems that the gap between C++ and Java has narrowed further since then. Important differences remain in numerical applications, also partly because C++, other than the Java, allows operator overloading.² However, MATSim's agent-based approach means that complex objects are handled much more frequently than true numerical computations.

- One reason for using C++ was that it could be combined with MPI, which is a reliable message passing standard for parallel computing. Parallel computing was necessary both for performance reasons and to be able to run simulations that needed more than about 4 GB of memory—the maximum that could be addressed with the 32 bit architecture standard at that time. MPI is also available for Java, but it is much less well maintained.

With the advent of the 64 bit architectures, the second reason for parallel computing became obsolete. In addition, with Kai Nagel now at a transport engineering department, it seemed that making conceptual progress was more important than keeping the parallel computing edge, especially since the maintenance of parallel code *permanently* consumes additional resources.

With the decision to give up on parallel computing, it was no longer necessary to maintain compatibility with MPI; thus, the move to Java was facilitated.

In terms of language, C# might have been an alternative to Java. However, C# depends much more on the Microsoft Windows platform, and community support is not as good as it is for Java.

Clearly, the code by Michael Balmer already had all the necessary data classes, readers and writers. The code was used as a starting point to re-implement MATSim in Java. Nevertheless, many important elements like mobsim, events architecture, scoring, routing, and co-evolutionary architecture had to be re-implemented. It took about two years from making that decision to the first plausible run of MATSim in Java.

Important early steps with MATSim in Java were to add time choice (Balmer et al., 2005b) and mode choice (Rieser et al., 2009) as additional choice dimensions beyond route choice. A summary of the status around 2008 was written by Balmer et al. (2009b).

46.2.1.5 Code Reorganization

The C++ version of MATSim was, similar to the original TRANSIMS, a collection of stand-alone executables coupled by scripts. For example, the router would read plans and events and replace some of the plans by other plans with modified routes. The program flow was organized with shell scripts and makefiles. Later, it was possible to start all modules simultaneously where they used messages to interact (also see Gloor and Nagel, 2005), but the file-based and scripted interaction always remained available.

² See http://en.wikipedia.org/wiki/Operator_overloading.

That approach had, in consequence, very clearly defined interfaces, i.e., the files. Exchanging information not included in the files meant changing the readers and writers on *both* sides, which was, in consequence, rarely done; stand-alone modules instead tried to work with the information they had.

When MATSim was re-implemented in Java around 2006/07, it was re-implemented as one system. Now, everything could interact with everything. For example, a router could modify the network, compute routes on the modified network and then modify it back. Clearly, it could make an error in the process, thus erroneously modifying the network. In this way, any module could modify any data of MATSim, greatly increasing the scope for misunderstandings and errors.

What created even more problems, however, were extensions to the program flow. The program flow was, as it still is, organized by the Controller class.³ Originally, everybody who wanted to change the program flow and insert his or her own research modules, would inherit from Controller, override some methods and insert his or her own instructions. This however, meant that it was impossible to *combine* the extensions without possibly massive manual interventions, illustrated as follows.

For example, assume the core program flow as

```
class Controller {
    void run() {
        ...
        aMethod() ;
        ...
    }
    void aMethod() {
        doA() ;
        doB() ;
    }
}
```

Also assume an extension called MyController from one researcher and another extension called YourController by another researcher:

```
class MyController extends Controller {
    @Override
    aMethod() {
        doA() ;
        doMyStuff() ;
        doB() ;
    }
}
```

```
class YourController extends Controller {
    @Override
    aMethod() {
        doA() ;
        doYourStuff() ;
        doB() ;
    }
}
```

If you wanted to combine both approaches, you could neither say YourController extends MyController nor MyController extends YourController, since either way one of the two extensions would get lost. In this simple case, one could possibly address the problem through manual

³ Mis-spelled since its inception.

intervention, but in more complicated situations this would no longer be possible without extensive additional testing.

Therefore, in 2008, a decision was made to make MATSim more modular. The first step in that direction was a decision to submit the whole MATSim repository to frequent refactorings, i.e., to *not* leave the code alone as much as possible, instead forcing the community to get used to frequent changes of code, while maintaining functionality. To facilitate that approach, coverage by automatic regression tests on the build server was hugely increased and all developers were encouraged to write automatic regression tests for their own code and projects.

The changes since then are too numerous to be listed here. They include, in particular, fairly restrictive data classes no longer extended or modified by every scientific project, and well-defined extension points in both the iterative loop and inside the mobsim. See Chapter 45 for currently existing extension points.

46.2.2 Kay W. Axhausen's Perspective

46.2.2.1 ORIENT/RV: Parking in Travel Demand Models (Karlsruhe University)

In 1984, Kay Axhausen returned to Karlsruhe University⁴ after two years doing an MSc degree at the University of Wisconsin, to start his PhD (Philosophiae Doctor – Doctor of Philosophy) at the IfV (Institut für Verkehrswesen/Institute for Transport Studies). At that time, the IfV already had a long tradition of traffic flow analysis (Leutzbach, 1972) and agent-based traffic flow simulation, as pioneered by Wiedemann (1974) (see also Leutzbach and Wiedemann, 1986). In this environment, Sparmann and Leutzbach (1980) had implemented a sample enumeration-based simulation of traffic demand in the spirit of Poock and Zumkeller (1978). This approach took the daily schedule of the traveler and simulated it activity-by-activity, including the necessary travel. Neither the traffic flow nor travel demand simulations aimed for equilibrium, but, in line with discussions at the time, both were more interested in the underlying behaviors (e.g., Jones et al., 1983).

Faced with a project to simulate parking as an extension of Sparmann's ORIENT approach, it became clear to Axhausen that sample enumeration approaches could not account for the temporal and spatial competition for parking spaces, but that the event-oriented approaches of the traffic flow model naturally could. Merging the two approaches was the natural solution and he then designed it for ORIENT/RV (Axhausen, 1989). Given the need to model the flow of traffic on the roads as part of the daily dynamics, the approach of Schwerdtfeger, an IfV colleague, was a natural and computationally-efficient choice. Schwerdtfeger (1984) had developed a mesoscopic simulation of traffic flow, which retained the agent-resolution, but employed macroscopic link-performance functions to calculate link speeds.

The work of Swiderski (1983), a second IfV colleague, started Axhausen thinking about the need to account for the constraints imposed by travelers' mental maps. As a full implementation of a mental map is impossible, even with today's computers, he chose to condition travelers' route choices on their travel time expectations, which were based on shortest-paths over an initially empty network. The agents reconsidered their routes at every junction if the experienced travel time deviated beyond an adaptive threshold from expected travel times. In this case, the route was recalculated with the current speeds. The framework was used to iterate (Axhausen, 1990) the expectations via shortest-paths based on stored mean travel times from the last iteration, but no formal tests of equilibrium were conducted, nor was the number of iterations extensive.

In the MATSim context, the competition for facilities was taken up by Horni et al. (2009). Reconsidering routing decisions while already being en-route was taken up by Dobler (2013),

⁴ Now: Karlsruhe Institute of Technology (KIT).

Number and type of activities

Sequence of activities

- Start and duration of activity
- Composition of the group undertaking the activity
- Expenditure division
- Location of the activity
- Movement between the sequential locations
 - Location of access and egress from the mean of transport
 - Parking type
 - Vehicle/means of transport
 - Route/service
 - Group traveling
 - Expenditure division

Source: Axhausen (2014, 2006, 2009)

Figure 46.1: Behavioral dimensions to be included in a fuller scheduling model.

where he showed that such an approach can approximate the equilibrium in a small number of iterations.

46.2.2.2 *From EUROTOPP to MATSim (Karlsruhe, Oxford, Innsbruck, Zürich)*

The first framework program of the European Union offered a chance to continue with the work in a larger context; unfortunately, this extended version of ORIENT/RV never went beyond the design stage (Axhausen and Goodwin, 1991). The EUROTOPP approach was later implemented in a changed form at the IfV, again by Zumkeller, who also had been one of the partners of the first framework project (Schnittger and Zumkeller, 2004), and his students.

Moving to Oxford, London, Innsbruck and then Zürich in rapid succession kept Axhausen from initiating serious work on a large-scale simulation system. The focus switched to data collection and choice modeling and collaboration on travel demand simulation with Kai Nagel began when he also joined ETH in 1999. While this was initially low key, Michael Balmer and David Charypar's move to Kay Axhausen's group after Kai Nagel's departure to TU Berlin jump-started further work, which is now documented in this book.

46.2.2.3 *"Best Response" and Further Choice Dimensions (ETH Zürich Transport Engineering)*

Departure time, mode and route choice are the heart of the transport modeling enterprise and were addressed in MATSim almost from the start (Raney and Nagel, 2004; Balmer et al., 2005b; Rieser et al., 2009). Work in Zürich addressed further behavioral dimensions, as shown in Figure 46.1. Each or past studies, which did not produce stable enough code for general use. It is clear that there are more dimensions to consider. Those listed in the figure are only the more obvious examples: for example, rail travel service class or activity engagement intensity are not addressed.

Today, MATSim takes the activity chain and schedule, as given from the initial demand generation process, as input; modern "activity based-models" make it sensitive to accessibility, understood

as the logsum term of the included destination and mode choice model (route choice is generally excluded in those models) (see Ben-Akiva et al., 1996, for an early example). Computational overhead costs of calculating non-chosen alternatives sets becomes prohibitive at the scale for which MATSim is designed, so alternative approaches were explored. Meister developed a **genetic algorithm on a household-basis to find optimal schedules for all members simultaneously** (reported in Meister et al., 2005), but only its time-of-day choice element was used in later scenarios (Meister et al., 2006). Feil set about finding a **best-response, but computationally fast approach to the optimization of the number and sequence of activities into a schedule** (Feil, 2010). While he made substantial progress using a tabu search and a cloning approach, it is still too slow as it currently stands. Fourie's PSim (see Chapter 39) might remove that constraint.

While Meister and of Feil's approaches, as well as the standard MATSim routing algorithm, attempt to directly provide best response solutions, the standard MATSim evolutionary algorithm also moves in the direction of good or best response (also see Section 97.3.1). With these approaches, it is impossible to directly model **destination choice**, since the best response destination would just be the closest possible destination (Horni et al., 2009). The problem: destinations similar from the analyst's point of view are quite different from each person's point of view: for example, allowing different types of leisure activity. As further explained in Chapter 27, the problem was addressed by attaching randomness directly to each person-alternative-pair (also see Horni et al., 2012b).

The need to address **parking** is obvious and even more so when considering electric vehicles and their current need to be recharged during the course of a day. Waraich addressed both aspects by integrating a local search into the overall MATSim iteration scheme to identify preferred parking spaces near the final destination (Chapter 13). Dobler's approach (Dobler, 2013) to evacuation is similar, but does not iterate, since that is not relevant for evacuation modeling. Waraich's local search can be extended with personalized walking time values.

The **group composition for joint travel and joint activities** is essential for making progress on a number of fronts, but especially to understand destination choice and activity generation. Gliebe and Koppelman (2005) or Zhang et al. (2005), for example, have proposed discrete choice models for household activity allocation. However, these approaches cannot be easily integrated into MATSim because of their computational costs. They are also too restrictive, with their exclusive focus on the household. Based on parallel empirical work on social networks (see Larsen et al., 2006; Kowald et al., 2013), Dubernet is currently exploring new game theoretic approaches to co-ordinate the timings and activities of households and wider social networks. These social networks are generated using the approach of Arentze et al. (2013), which was estimated against Swiss data for leisure social contact (Kowald and Axhausen, 2012) so as to reproduce measured characteristics of the real network, such as homophily, clustering and average number of leisure social contacts.

The **expenditure division** question is a promising research avenue (Section 97.6) not yet explored by transport planning and clearly interacting with joint activity participation and travel.

CHAPTER 47

Agent-Based Traffic Assignment

Kai Nagel and Gunnar Flötteröd

47.1 Introduction

This chapter presents MATSim from a DTA perspective. The following material is an abridged and edited version of Nagel and Flötteröd (2012).

The traffic assignment problem, whether macroscopic or microscopic, static or dynamic, trip-based or agent-based, is to identify a situation where travel demand and travel supply (network conditions) are consistent with each other. Travel demand results from a demand model that reacts to conditions in the network; these are the output of a supply model (network loading model) using travel demand as its input. A solution of the traffic assignment problem describes an equilibrium between travel demand and travel supply.

Possibly, the most intuitive mathematical formulation of this problem is defined by a fixed point: Find a demand pattern generating network conditions that, in turn, cause the same demand pattern to re-appear. This formulation is operationally important because it motivates a straightforward way of calculating an equilibrium by alternately evaluating the demand model and the supply model. If these iterations stabilize, a fixed point is attained that solves the traffic assignment problem.

The remainder of this chapter places MATSim into this DTA framework. Section 47.2 starts out from the static and macroscopic assignment of route flows and incrementally enriches this formulation into a dynamic and fully disaggregate agent-based assignment problem. Section 47.3 then turns to the problem of how to simulate (solve) this model system, with a particular focus on MATSim's coevolutionary approach. Section 47.4 concludes the presentation.

How to cite this book chapter:

Nagel, K and Flötteröd, G. 2016. Agent-Based Traffic Assignment. In: Horni, A, Nagel, K and Axhausen, K W. (eds.) *The Multi-Agent Transport Simulation MATSim*, Pp. 315–326. London: Ubiquity Press. DOI: <http://dx.doi.org/10.5334/baw.47>. License: CC-BY 4.0

47.2 From Route Swapping to Agent Plan Choice

The following details an increasingly comprehensive specification of the traffic assignment problem, starting from the classical static user equilibrium model and ending with a fully dynamic model that captures arbitrary travel demand dimensions at the individual level. Computationally, the iterative fixed point solution procedure is carried throughout the entire development. Deliberately, this solution method also has a behavioral interpretation as a model of day-to-day replanning; see also Section 97.3.5.

We start by considering route assignment only. The generalization towards further choice dimensions will turn out to be rather straightforward.

47.2.1 Static Traffic Assignment

Consider a network of nodes and links, where some, or all, of the nodes are demand origins, denoted by o , and/or demand destinations, denoted by d . The constant demand q^{od} in an O-D relation od splits up among a set of routes K^{od} . Denote the flow on route $k \in K^{od}$ by r_k^{od} , where $\sum_{k \in K^{od}} r_k^{od} = q^{od}$.

Most route assignment models either specify a UE (User Equilibrium a.k.a. Wardrop's first principle) or an SUE (Stochastic User Equilibrium). A UE postulates that r_k^{od} is zero for every route k of non-minimal cost (Wardrop, 1952):

$$c(k) = \min_{s \in K^{od}} c(s) \Rightarrow r_k^{od} \geq 0 \quad (47.1)$$

$$c(k) > \min_{s \in K^{od}} c(s) \Rightarrow r_k^{od} = 0 \quad (47.2)$$

where $c(k)$ is the cost (typically delay) on route k .

An alternative, frequently-used approach is to distribute the demand onto the routes such that an SUE is achieved, where users have different perceptions of route cost and every user takes the route of *perceived* minimal cost (Daganzo and Sheffi, 1977). Mathematically, this means that the route flows fulfill some distribution

$$r_k^{od} = P_k^{od}(c(x(\{r_k^{od}\}))) \cdot q^{od} \quad (47.3)$$

where the route splits P_k^{od} are a function of the network costs $c(x)$, which depend on the network conditions x , which, in turn, depend on all route flows $\{r_k^{od}\}$.

In either case, the model needs to be solved iteratively, which typically involves the following steps (Sheffi, 1985):

Algorithm 47.1 Macroscopic and static route assignment

1. **Initial conditions:** Compute some initial routes (e.g., best path on empty network for every O-D pair).
 2. **Iterations:** Repeat the following many times.
 - (a) **Network loading:** Load the demand on the network along its routes and obtain network delays (congestion).
 - (b) **Choice set generation:** Compute new routes based on the network delays.
 - (c) **Choice:** Distribute the demand between the routes based on the network delays.
-

Defining the network loading as more on the “physical” side of the system, the behaviorally relevant steps are choice set generation and choice (Bowman and Ben-Akiva, 1998).

Choice set generation: Often, the new routes are best paths based on the last iteration (“best reply” or “best response” choice set generation). The routes are generated within the iterations because an a priori enumeration of all possible routes is computationally unfeasible.

Choice: Usually, demand is shifted among the routes to improve consistency with the route choice model, assuming—in the simplest case—constant network delays: In a UE, the flow on the current best routes is increased at the cost of the other route flows (“best reply” or “best response” choice), whereas for an SUE, flows are shifted towards the desired route choice distribution (often a version of multinomial logit, e.g., Dial, 1971; Cascetta et al., 1996; Ben-Akiva and Bierlaire, 1999). For stability reasons, this shift is typically realized in a gradual way that dampens the iteration dynamics. See below for more discussion on convergence issues.

The **iterations** are repeated until some stopping criterion is fulfilled, indicating that a fixed point is attained. In the best reply situation, the fixed point implies that no shift between routes takes place, i.e., what comes out as the best reply to the previous iteration is either the same, or at least of the same performance, as what was used in the previous iteration. Since, in this situation, no O-D pair can unilaterally improve by switching routes, the system is at a Nash equilibrium (e.g., Hofbauer and Sigmund, 1998). In the SUE situation, the fixed point means that a route flow pattern $\{r_k^{od}\}$ is found that leads to exactly those network conditions the travelers (the O-D flows) perceived when choosing their routes, giving no incentive to re-route.

Destination choice and elasticity in the demand are behavioral dimensions beyond route choice that can be captured by a static model. However, no technical generality is lost when discussing only route choice; both additional choice dimensions can be rephrased as generalized routing problems on an extended network (“supernetwork”; see, e.g., Sheffi, 1985; Nagurney and Dong, 2002).

47.2.2 Dynamic Traffic Assignment

The process above also works for *dynamic* traffic assignment (DTA; see Peeta and Ziliaskopoulos, 2001), where both demand and network conditions are time-dependent and the time-dependent travel times in the network define a physically meaningful progression of a demand unit through the network.

The algorithm structure does not change. The individual steps now look as follows:

Algorithm 47.2 Macroscopic and dynamic route assignment

1. **Initial conditions:** Compute some initial routing (e.g., best path on empty network for every O-D pair and departure time).
 2. **Iterations:** Repeat the following many times.
 - (a) **Network loading:** Load all demand items on the network according to their departure times, let them follow their routes and obtain network delays (congestion).
 - (b) **Choice set generation:** Compute new routes based on the network delays.
 - (c) **Choice:** Distribute the demand between the routes based on the network delays.
-

Once more, *if* the new routes are best replies (i.e., best paths based on the last iteration), *if* demand is shifted towards these new routes and *if* these iterations reach a fixed point, then this is then a dynamic UE since best reply dynamics mean that no traveler (no O-D flow) can unilaterally deviate to a better route. The SUE interpretation carries over in a similar way.

Destination choice and elasticity in demand apply naturally to the dynamic case as well. Beyond this, the dynamic setting also enables the modeling of departure time choice. Again, the sole consideration of route choice does, at least technically, not constitute a limitation because departure

time choice can be translated into route choice in a time-expanded version of the original network (van der Zijpp and Lindveld, 2001).

47.2.3 Individual Travelers

In both the static and dynamic case, it is possible to re-interpret the algorithm in terms of individual travelers. In the static case, for every O-D pair, one needs to assume a steady (= constant) flow of travelers entering the network at the origin at a constant rate, corresponding to that O-D flow. A solution to the static assignment problem corresponds to the distribution of different travelers onto possibly different paths.

In the dynamic case, one needs to generate the appropriate number of travelers for every O-D pair and every time slot and distribute them across the time slot. From then on, the triple (origin, destination, departure time) is fixed for every simulated traveler; its goal is to find an appropriate path. Arguably, this re-interpretation is behaviorally more plausible in the dynamic case.

In a trip-based context, there are two major motivations to go from continuous flows to individual travelers:

- Traffic flow dynamics in complex network infrastructures are difficult to model as continuous flows (e.g., Flötteröd and Rohde, 2011), but are relatively straightforward to simulate at the individual vehicle level (TSS Transport Simulation Systems, accessed 2015; Quadstone Paramics Ltd., accessed 2015; Caliper, accessed 2015; PTV AG, accessed 2015; DynusT, accessed August 2014; Zhou and Taylor, 2014). Disaggregating an O-D matrix into individual trip-makers allows the assignment of one vehicle to every trip-maker in the microscopic traffic flow simulation.
- It is computationally inefficient to capture demand heterogeneity through a large number of commodity flows, but the sampling of trip-makers with different characteristics is fairly straightforward. For example, every vehicle can be given an individual route to its individual destination.

For a finite population of heterogeneous travelers, each traveler constitutes an integer commodity and the **choice** step thus must be changed from “gradually shift the route flows towards something consistent with the behavioral model” into “for a *fraction* of travelers, assign a *single* behaviorally plausible route to each of these travelers”. The gradual shift that helps stabilize iterations in the continuous assignment carries over here to an equally stabilizing “inert shift”; not all travelers change their routes at once. This is a consistent reformulation; if one reduces the traveler size from one to $\varepsilon \rightarrow 0$ and increases the number of travelers by a factor of $1/\varepsilon$, a 10 % chance of changing routes in the disaggregate case carries over to shifting 10 % of all flows to new routes in the aggregate case (“**continuous limit**”).

Apart from this, the iterations do not look much different from what was shown before:

Algorithm 47.3 Microscopic and dynamic route assignment

1. **Initial conditions:** Compute some initial routing (e.g., best path on empty network for every traveler).
 2. **Iterations:** Repeat the following many times.
 - (a) **Network loading:** Load all travelers on the network according to their departure times, let them follow their routes and obtain network delays (congestion).
 - (b) **Choice set generation:** Compute new routes based on the network delays.
 - (c) **Choice:** Assign every traveler to a route (which can be the previously chosen one) based on network delays.
-

UE and SUE notions carry over to the disaggregate case if the notion of an O-D pair (or a commodity) is replaced by an individual **particle** (= microscopic traveler).

A **particle UE** may be defined as a system state where no particle can unilaterally improve itself. This definition is consistent with definitions in game theory, which normally start from the discrete problem. It should be noted, however, that this makes the problem combinatorial, which means that even a problem that had a unique solution in its continuous version may have a large number of solutions in its discrete version. Further, the complexity of finding a solution increases similarly to the situation where linear programming jumps to being NP-hard when the variables are required to be integers.¹ The particle UE is hence deliberately not searching for an integer approximation of the continuous solution.

Situations may occur where mixed strategy equilibria exist; these are equilibria, where participants draw between different fixed strategies randomly. This implies that the opponents need to interpret the outcome of the game probabilistically: Even if they themselves play fixed strategies, they need to maximize some expectation value.

For a **particle SUE**, the continuous limit assumption of the macroscopic model is discarded in that the choice fractions $P_k^{od}(c(x(\{r_k^{od}\})))$ in (47.3) are now interpreted as individual-level choice probabilities $P_k^n(c(x(\{r_k^n\})))$ where r_k^n is a binary variable that indicates if traveler n takes route k or not. This implies that the individual-level route flows r_k^n are now random 0/1 variables; consequently, the cost structure—based on individual choices made—becomes probabilistic as well (Balijepalli et al., 2007; Cascetta and Cantarella, 1991; Cascetta, 1989).

A particle SUE is defined as a system state where travelers draw routes from a stationary choice distribution and where the resulting distribution of traffic conditions re-generates that choice distribution.

An operational **particle SUE** specification results if one assumes that travelers filter out the random fluctuations from what they observe and base their decisions only on average route costs:

$$P_n(k) = P_n(k | E\{c(x(\{r_k^n\}))\}) \quad (47.4)$$

where $P_n(k)$ now is the probability that trip-maker n selects route k and $E\{\cdot\}$ denotes the expectation. This approach incorporates some generality; it can be shown that choice distributions based on expected network conditions coincide, up to first order, with the stationary choice distributions based on fluctuating network conditions (Flötteröd et al., 2011).

The resulting route flows r_k^n represent not only mean network conditions but also their variability, due to the individual-level route sampling. Alternatively, one could use the particles merely as a discretization scheme of continuous O-D flows and distribute them as closely as possible to the macroscopic average flow rates (e.g., Zhang et al., 2008). The latter approach, however, does not lend itself to the subsequently developed behavioral model type.

No new behavioral dimensions are added when going from commodity flows to particles. However, the microscopic approach allows simulation of greater behavioral variability within the given choice dimensions because it circumvents the computational difficulties of tracking a large number of commodity flows. This will be discussed in more detail in Section 47.2.5.

47.2.4 Stochastic Network Loading

The network loading can be deterministic or stochastic. With deterministic network loading, given time-dependent route inflows, one obtains one corresponding vector of network costs. With stochastic network loading, given the same input, one obtains a *distribution* of vectors of network costs.

¹ See http://en.wikipedia.org/wiki/Linear_programming_relaxation.

The macroscopic SUE approach of Section 47.2.1 assumes a distribution of choices but converts choice probabilities into choice fractions before starting the network loading. That is, one effectively performs $\text{NetworkLoading}(E\{\text{Choices}\})$. It is, however, not clear that this is the same as $E\{\text{NetworkLoading}(\text{Choices})\}$; in fact, with a non-linear network loading, even when it is deterministic, the two are different (Cascetta, 1989). Any Monte Carlo simulation of the particle SUE makes this problem explicit: If, at the choice level, one generates draws from the choice distribution, it makes sense to *first* perform the network loading and *then* do the averaging, rather than the other way around. This is especially true if day-to-day replanning is modeled, in which case draws from the choice distribution have a behavioral interpretation as the actual choices of the trip makers on a given day (but see also Section 97.3.5).

This, however, makes the output from the network loading effectively stochastic since the input to the network loading is stochastic. In consequence, any behavioral model that uses the traffic conditions as input needs to deal with the issue that these inputs are stochastic. *Thus, using a stochastic instead of a deterministic network loading makes little difference.* Making the network loading stochastic simplifies the implementation of certain network loading models. In particular, randomness is a method to resolve fractional behavior in a model with discrete particles.

With stochastic network loading, additional aspects of the iterative dynamics need to be defined. For example, a “best reply” could be against the last stochastic realization or against some average.

47.2.5 Extending the Route Assignment Loop to Other Choice Dimensions

Given the above behavioral interpretation, it is now straightforward to extend the assignment loop to other choice dimensions. For example, the “best reply” can include optimal departure time choice (e.g., de Palma and Marchal, 2002; Ettema et al., 2003) or optimal mode choice. This becomes easiest to interpret (and, in our view, most powerful in practice) if one moves from the concept of “trips” to daily plans. MATSim plans maintain the structure of DTA in terms of the triple (origin, departure time, destination); see Section 2.2.2.3 for an example. However, different from DTA, all activities are chained together.

This widens the behavioral modeling scope dramatically; all choice dimensions of an all-day travel plan can now be jointly equilibrated. This increases the degrees of freedom that need to be modeled but also carries a set of natural constraints along, which again reduce the solution space. Most notably, the destination of one trip must be the origin of the same agent’s (synthetic traveler’s) subsequent trip and an agent must arrive before it departs. Also, constraints such as Hägerstrand’s space-time prisms (Hägerstrand, 1970) are automatically enforced when the agents need to return to their starting locations.

There is so significant conceptual difference between the network loading of a route-based and a plan-based model.

The notion of a particle (S)UE can now be naturally extended to agents that execute complete plans.

An **agent-based UE** implies individual travelers (Section 47.2.3), additional choice dimensions (Section 47.2.5) and possible stochastic network loading (Section 47.2.4). Corresponding to the particle UE, it is defined as a system state where no agent can unilaterally improve its plan.

An **agent-based SUE** implies individual travelers (Section 47.2.3), additional choice dimensions (Section 47.2.5) and, normally, stochastic network loading (Section 47.2.4). Corresponding to the particle SUE, it is defined as a system state where agents draw from a stationary choice distribution and where the resulting distribution of traffic conditions re-generates that choice distribution.

If the iterations aim at an agent-based UE, then choice set generation and choice should implement a “best reply” logic; some ‘optimal’ plans are calculated and assigned to the agents. This is anything but an easy task.

The disaggregate counterpiece of an SUE implies that every agent considers a whole choice set of (possibly suboptimal) plans and selects one of these plans probabilistically, which can lead to huge data structures.

Summarizing, we have now arrived at a fully disaggregate dynamic DTA specification that accounts for arbitrary behavioral dimensions.

47.3 Agent-Based Simulation

The conceptual validity of the agent-based traffic assignment model is fairly intuitive. However, since it comes with a substantial computational burden of solving the model, it presents entirely new challenges on the simulation side.

On the demand side, there is, in particular, the combinatorial number of choice alternatives that must be considered. For example, random utility models rely on an a-priori enumeration of a choice set that represents options each traveler considers when making a choice (Ben-Akiva and Lerman, 1985). This choice set is huge in an agent-based simulation (Bowman and Ben-Akiva, 1998). While there are sampling-based approaches to the modeling of large choice sets that aim at reducing this computational burden, they have not yet been carried over to the modeling of all-day-plan choices (Ben-Akiva and Lerman, 1985; Frejinger et al., 2009b; Flötteröd and Bierlaire, 2013). See also Chapter 49.

As long as household interactions are not included, the demand modeling problem can be decomposed by agent once the network conditions are given—a great computational advantage. The supply model, on the other hand, deals with congestion, which is, by definition, a result of all travelers' physical interactions. Modeling large urban areas requires dealing with millions of travelers, and an operational supply simulation must be able to load all of these travelers on the network in reasonable computation time.

The following sections describe solutions for these problems. Concrete examples of much of this material are implemented within MATSim.

47.3.1 Agent-Based UE; One Plan per Traveler

The simulation of an agent-based UE is possible through the following implementation of the behavioral elements.

Choice set generation: For every agent, generate what would have been best in the previous iteration. This does not concern just the route but all considered choice dimensions, e.g., departure times and/or mode choice.

Choice: Switch to the new plan with a certain probability.

The choice set generation implements a “best reply” dynamic. This now requires identification of an optimal all-day plan for given network conditions. While the calculation of time-dependent shortest paths for UE route assignment is computationally manageable, the identification of optimal plans is far more difficult (Recker, 2001). This is an important technical motivation to switch to an agent-based SUE, where optimality is not required (see below).

Even in the manageable cases of, e.g., shortest paths, any best reply computation is an approximation. Time-dependent routing algorithms require knowledge of every link's travel time as a function of the link entrance time. In computational practice, this information exists only in an average, interpolated way. Thus, such computations become more robust if plan performance is directly taken from the network loading instead of relying on the best reply computation prediction; an agent sticks with a new plan only if it *performs* better than the previous plan (Raney and Nagel, 2004). However, to keep run times manageable in computational practice, multiple agents

must make such trial-and-error moves simultaneously. This is, therefore, not an exact best reply algorithm.

For the choice, a useful approach is to make the switching probability from current to best reply solution proportional to the expected improvement, i.e.,

$$P(\text{old} \rightarrow \text{best}) = \min[1, \mu \cdot (S_{\text{best}} - S_{\text{old}})]$$

where S_{best} and S_{old} are the (expected) scores of the best reply and the old plan, respectively, and the min takes care of the fact that a probability should not be larger than one. Truncation at zero is not necessary because the term $S_{\text{best}} - S_{\text{old}}$ cannot become negative. Chapter 3 gives an example of what a scoring function for all-day plans could look like. Note how the decreasing switching *fraction* of the continuous case is replaced by a decreasing switching *probability* (leading to a switching *rate*).

Clearly, any fixed point of such iterations is a UE since no switching takes place at the fixed point, meaning that the best reply plan has the same score as the already existing plan. Stability of the fixed point depends on the switching rate slope at the fixed point, in the above formulation on the μ : All else equal, making μ smaller makes the fixed point more stable but slows down convergence. These observations hold not only in transportation (e.g., Watling and Hazelton, 2003) but quite generally in the area of “evolutionary games and dynamical systems” (Hofbauer and Sigmund, 1998). In addition, in the context of traffic assignment, the existence of physical queues allowing for spillback across many links has apparently been shown to be an inevitable source of multiple Nash equilibria (Daganzo, 1998).

Alternatively, some MSA-like scheme may be used (Liu et al., 2007). One disadvantage with MSA is that the switching rate does not depend on the magnitude of the expected improvement, which could mean slow(er) convergence. An advantage of MSA is that one does not need to find out a good value for the proportionality factor (μ in the above example).

Yet another approach would be to use a “gap” function measuring the distance of the current assignment from an equilibrium and to infer the switching rate from the requirement that this function must be minimized (Lu et al., 2009; Zhang et al., 2008). However, we are not aware of any operational gap function that applies to all-day plans.

The biggest criticism of agent-based UE is its lack of behavioral realism. In a UE, every agent is assumed to react with a best response according to a model of its objectives, which implies that real travelers are able to compute best responses despite their combinatorial nature and high dimension (Bowman and Ben-Akiva, 1998). Furthermore, as in a pure route assignment, it is reasonable to assume that (i) the behavioral objective is imperfectly modeled and that (ii) explorative travel behavior leads to—more or less—random variations in what real travelers do. While (ii) explicitly introduces stochasticity, (i) calls for it as a representation of imprecision in the behavioral model.

These considerations do not only lead naturally to the agent-based SUE; they also stimulate an additional behavioral component capturing real travelers’ explorative learning. Similar to the symmetry between day-to-day replanning and the traffic assignment problem’s iterative solution, an explorative learning algorithm can be interpreted either as a model of real learning or as a computational method to solve a stochastic assignment problem. The following section presents a possible implementation of such an algorithm.

47.3.2 Agent-Based SUE; Multiple Plans per Traveler

This section discusses MATSim’s co-evolutionary algorithm for simulating plan choices. Chapters 49 and 51 provide an alternative perspective on MATSim’s plan choice mechanisms in terms of mainstream discrete choice theory (Ben-Akiva and Lerman, 1985).

It is possible to approach every agent's daily planning problem as a population-based search algorithm. Such a search algorithm maintains a *collection* (= population) of possible solutions to a problem instance and obtains better solutions via that collection's evolution. This is a typical machine-learning (e.g., Russel and Norvig, 2010) approach; the best-known population-based search algorithms (also called **evolutionary algorithms**) are **genetic algorithms** (e.g., Goldberg, 1989).

It is important to note that “population” here refers to the collection of solutions for a single individual, as opposed to the population of travelers. Every individual uses a population-based algorithm to “co-evolve” in the population of all travelers (also see Balmer, 2007).

A **population-based search algorithm** typically works as follows:

Algorithm 47.4 Population-based search

1. **Initiation:** Generate a collection of candidate solutions for a problem instance.
 2. **Iterations:** Repeat the following many times.
 - (a) **Scoring:** Evaluate every candidate solution's “score” or “fitness”.
 - (b) **Selection:** Decrease the occurrence of “bad” solutions. There are many ways how this can be done.
 - (c) **Construction of new solutions:** Construct new solutions and add them to the candidate solutions collection.
-

For the construction of new solutions, two operators are often used in genetic algorithms: **Mutation**—which takes a candidate solution and performs small modifications to it; and **crossover**—which takes *two* candidate solutions and constructs a new one from those. Since mutation takes one existing solution and crossover takes two, it makes sense to also move in the opposite direction and define an operator that takes zero solutions as input, i.e., generates **solutions from scratch**—a “best-reply to last iteration” would, for example, be such an operator.

Like what has been said before, we typically have a situation where multiple travelers evolve simultaneously: a *population of persons* where every person has a *population of plans*. The result is a co-evolutionary dynamic, where each person evolves according to a population-based **co-evolutionary algorithm**. The overall approach reads as follows (see, e.g., Hrabner et al., 1994; Arthur, 1994, for similar approaches):

Algorithm 47.5 Co-evolutionary, population-based search

1. **Initiation:** Generate at least one plan for every agent.
 2. **Iterations:** Repeat the following many times.
 - (a) **Selection/Choice:** Select one of the plans for every agent.
 - (b) **Scoring:** Obtain a score for every agent's selected plan by executing all selected plans simultaneously in a simulation and attaching some performance measure to each executed plan. Clearly, what was previously the network loading has now evolved into a full-fledged agent-based simulation of daily activities. See Section 47.3.2.4 for more detail on scoring.
 - (c) **Generation of new plans (innovation)/Choice set generation:** For some of the agents, generate new plans; for example, as “best replies” or as mutations of existing plans (e.g., small departure time changes).
-

Note that this approach is really quite congruent with the SUE approach: Each person has a plan collection, which may be interpreted as the choice set. As in SUE, the choice set can be generated while the iterations run or before the iterations start. Each person selects between the plans, where one can attach to every plan a score-based probability to be selected, which becomes in the end similar to Equation (47.3). Clearly, a relevant related research topic is to specify an evolutionary dynamic that can be shown to converge to choice sets that are generated consistently with discrete choice theory requirements; see Chapter 49 and Section 97.3.

The following subsections give examples for the different elements of this approach.

47.3.2.1 Selection (Choice)

A possible choice algorithm is the following: For persons with unscored plans, select an unscored plan. For all other persons, select between existing plans with some SUE model, e.g., a logit model, i.e.,

$$P(i) = \frac{e^{\mu S_i}}{\sum_j e^{\mu S_j}} \quad (47.5)$$

where S_i is the score of plan i and μ models the travelers' ability to distinguish between plans with different scores. This is implemented in MATSim by `SelectExpBeta`.

In practice, we have found that it is much better to not use Equation (47.5) directly but instead use a switching process that *converges* towards Equation (47.5). This can, for example, be achieved by using a switching probability from i to j of the form

$$T(i \rightarrow j) = \gamma e^{\beta(S_j - S_i)/2} \quad (47.6)$$

where i is the previous plan, j is a randomly selected plan from the same person and γ is a proportionality constant that needs to be small enough so that the expression is never larger than one (since it denotes a probability). This works because the logit model (47.5) fulfills the detailed balance condition

$$P(i) T(i \rightarrow j) = P(j) T(j \rightarrow i) \quad (47.7)$$

for these $T(i \rightarrow j)$ (e.g., Ross, 2006).² This is implemented in MATSim by `ChangeExpBeta`.

The “switching approach” has additional advantages, including the following:

- Equation (47.6) can be behaviorally interpreted as the probability of switching from plan i to plan j . Plausibly, this probability increases with the magnitude of the improvement. For certain applications, one might want a more involved approach, e.g., an *expected* score of j , which then initiates the switch.
- One could replace Equation (47.6) by a threshold-based dynamics, i.e., a switch to a better solution will only take place if the improvement is above a certain threshold. One loses some of the mathematical interpretation, but it may be more consistent with discussion of project appraisal, where small improvements may not lead to a change in behavior.

Although not performed systematically in past work, it is possible to include formulations such as path-size logit (Ben-Akiva and Bierlaire, 1999) in the choice model.

² Assume that, after a number of iterations, there is no more innovation, i.e., the choice set for every agent is fixed and that the scores are updated by MSA. On convergence of the iterations, all agents draw their plans from a fixed choice set based on constant score expectations, cf. (47.4). This means that all agents make their choices independently (and that all interactions are captured in the scores). The switching logic (47.6) then defines an ergodic Markovian process, which converges to the unique steady state probabilities (47.5).

47.3.2.2 Score Convergence

The assumption that the scores eventually converge to some constant value intuitively means that the scores cannot display spontaneous reactive behavior to a certain iteration. For example, a particular iteration might display a “network breakdown” (Rieser and Nagel, 2008). Converged scores would not trigger a next-day reaction to that breakdown. In practice, this can be achieved by averaging the scores over many iterations, which is somewhat similar to fictitious play (Monderer and Shapley, 1996; Garcia et al., 2000). Once more, MSA is an option (Section 3.3.4), with the same advantages and disadvantages discussed before (Section 47.3.1). An alternative is to use a small **learning rate** α (Section 3.3.3) in

$$S_i^{\text{new}} = (1 - \alpha)S_i^{\text{old}} + \alpha \tilde{S}_i \quad (47.8)$$

where S_i^{new} and S_i^{old} are the agent’s memorized scores for option i , and \tilde{S}_i is the most recent actual performance with that option; also see Chapter 49. The issue, in the end, is the same as the stable-vs-unstable fixed points (cf. Section 47.3.1): If the system is well-behaved (corresponding to a stable fixed point), it will converge benignly to constant scores and thus to the detailed balance solution. If the system is not well-behaved, one can still force it to such a solution with MSA, but the meaning is less clear (also see Sections 3.3.4 and 47.3.2.2).

As stated before, stochastic network loading makes no additional conceptual difference since there is already stochasticity caused by choice behavior.

47.3.2.3 Innovation (Choice Set Generation)

So far, this leaves open the question on choice set *generation*, i.e., the part that generates new plans or modifies existing ones.

One computationally simple technique not requiring a choice set enumeration is to simulate randomly disturbed link costs and run best response based on these costs. This, however, can yield unrealistic results if one does not get the correlation structure of the noise right.

An alternative is to calculate separate best responses after every network loading. Since the process is stochastic, this will generate different solutions from iteration to iteration. An advantage is that the correlations will be generated by the simulation—and are, presumably, realistic. Chapter 49 relates this to random utility modeling; see also Chapter 97.

Beyond that, there are many different algorithms that could be used here. Besides the previously-mentioned “mutation” (Balmer et al., 2005b) or “crossover” (Charypar and Nagel, 2005; Meister et al., 2006), there are also many possibilities for constructive algorithms, such as “agent-based” construction (Zhu et al., 2008). One attractive option, clearly, is to use a regular activity-based demand generation code (e.g., Bowman et al., 1998; Miller and Roorda, 2003), although we have found that this may not be as simple as it seems (Rieser et al., 2007b); in practice, activity-based models are often constructed with O-D matrices in mind. A successful integration is described by Ziemke et al. (2015).

47.3.2.4 Adjusting the “Improvement Function” from Shortest Time to Generalized Utility Functions

This chapter takes an inductive approach and argues that one can make the network assignment loop more general by including additional choice dimensions beyond routing. Clearly, for this to work, the scoring needs to take the effects of these additional choice dimensions into account (also see Balmer, 2007). Given evolutionary game theory, it is quite obvious how to do that: One has to extend the cost function used for routing to a general scoring function for complete daily plans.

That is, the performance of a daily plan needs to be scored. An established method to estimate scoring functions for different alternatives is random utility theory (e.g. Ben-Akiva and Lerman, 1985), which is why in the following “scoring” will be replaced by “utility”. For a utility function for daily plans, the following arguments may serve as starting points:

- A heuristic approach, consistent with wide-spread assumptions about travel behavior, is to give positive rewards to performing an activity and negative rewards to traveling.
- For the activities, one should select functions where the marginal reward of doing an activity decreases over time.
- Without additional effects, such as opening times or time-varying congestion, the marginal utilities of all performed activities should be the same.

MATSim has, in the past years, gained some experience with the approach described in Chapter 3 and with more theory in Chapter 51; this then closes the loop.

47.4 Conclusion

Starting from regular route assignment, this chapter explains how one can extend the iterative solution procedure of static or dynamic traffic assignment to include additional behavioral dimensions such as time adaptation, mode choice or secondary activity location choice. This is somewhat similar to the so-called supernetworks approach but argues from the viewpoint of the iterative solution procedure rather than the problem definition.

To address the combinatorial explosion of commodities caused by the expansion of the choice dimensions, a move to individual particles is suggested. This allows an interpretation of the solution procedure as behavioral day-to-day learning but maintains a connection to the SUE definition by interpreting synthetic travelers’ behavior as random draws from individual choice sets.

Most of this chapter discusses simulation/computer implementation issues. From the definition given above, progress can be made by using methods from machine learning and co-evolutionary search algorithms. The SUE problem of random selection between different alternatives can be cast as a so-called population-based optimization algorithm, where each synthetic traveler randomly selects between the different members of the population of possible solutions. At the same time, the *population of the travelers* co-evolves towards a stationary distribution of choices.

Overall, this chapter has worked out the structural similarity between the “classical” DTA problem and the more recent agent-based assignment problem.³ The presentation has focused on the algorithmic issue of how to find solutions to these problems. This is complemented by the subsequent Chapters 48 to 50, which mostly discuss modeling (descriptive) aspects of MATSim.

³ It is, in fact, possible to run MATSim in DTA mode, by converting each trip into a dummy person, with dummy activities at the beginning and end of the trip. The class `RunExample5Trips` (see <http://matsim.org/javadoc> → main distribution) runs an example; the class itself points to a configuration file, which in turn points to `examples/equil/plans100trips.xml`. A dummy person that denotes a trip from link 1 to link 20, departing at 6 am, is coded as

```
<person id="1">
  <plan>
    <act type="dummy" link="1" end_time="06:00" />
    <leg mode="car" />
    <act type="dummy" link="20" dur="00:10" />
  </plan>
</person>
```

CHAPTER 48

MATSim as a Monte-Carlo Engine

Gunnar Flötteröd

48.1 Introduction

“Agents” that “learn” in a “synthetic reality” is a common term in Artificial Intelligence (Russel and Norvig, 2010) and/or Multi-Agent simulation (Ferber, 1999), but it does not belong to the standard terminology of transport modeling. This chapter explains the functioning of MATSim in terms of modeling and simulation concepts that are more established in the transportation field.

It is important to distinguish between a model and a simulation. A model describes certain aspects of a system; a simulation evaluates a model. For instance, a simple route choice model may state that route A is selected with 25 % probability and route B with 75 % probability. A simulation of this model then draws one or more realizations (route choices) from this distribution. One always needs a model before one can simulate. Possible feedback from simulation to modeling comprises (i) new insights into emergent model properties and (ii) computational constraints that prohibit overly complex model specifications. In MATSim, both kinds of feedback are strong drivers of the modeling.

Consider Figure 48.1, displaying MATSim as a model system comprising a (travel) demand model and a (network) supply model. The travel demand model predicts travelers’ behavior, given their information about the network conditions. The network supply model predicts these network conditions using a certain travel behavior chosen by all travelers in the system. This is complemented by the modeling assumption that demand and supply are mutually consistent in the sense that the network conditions resulting from a certain travel behavior are statistically equal to the network conditions that caused this behavior.

Simulation addresses the question of how to identify this state of mutual demand/supply consistency, i.e., it solves the model. The model system shown in Figure 48.1 is complicated—it is nonlinear, stochastic and extremely high-dimensional. The only known operational technique to solve it exploits an additional modeling assumption that justifies the real occurrence of

How to cite this book chapter:

Flötteröd, G. 2016. MATSim as a Monte-Carlo Engine. In: Horni, A, Nagel, K and Axhausen, K W. (eds.) *The Multi-Agent Transport Simulation MATSim*, Pp. 327–336. London: Ubiquity Press. DOI: <http://dx.doi.org/10.5334/baw.48>. License: CC-BY 4.0

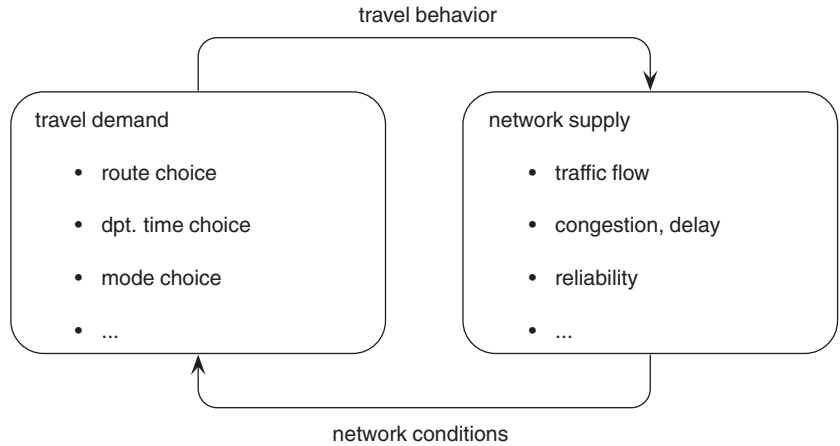


Figure 48.1: Demand/supply perspective on MATSim.

Algorithm 48.1 Iterative scheme to reach demand/supply consistency

1. Create a synthetic agent population.
 2. Create a synthetic environment.
 3. Iterate:
 - (a) All agents choose some planned travel behavior.
 - (b) All agents execute their travel plans.
 - (c) All agents see the resulting network conditions.
-

demand/supply consistency: travelers adjust their behavior for their own benefit and only stop doing that when further improvement is insubstantial. Demand/supply consistency characterizes the outcome of this process jointly for all travelers.

Now, consider Algorithm 48.1, which displays the high-level simulation logic of MATSim. This is indeed a logic that iteratively adjusts travel demand. If this logic adjusts the simulated behavior of the simulated travelers until further simulated improvements are insubstantial, then this logic should approach a state of demand/supply consistency. That is, Algorithm 48.1 may be a valid solution method for the model system shown in Figure 48.1. However, that model system does not specify how demand and supply become consistent; it merely specifies that this eventually happens. The only modeling assumption made is that some process of this type exists. The purpose of Algorithm 48.1 is not to mimic this (unspecified) process; it only identifies the final outcome of that process.

The fact that Algorithm 48.1 mimics real, urban, day-to-day dynamics invites misleading interpretations of the underlying model system. In particular, it is a misconception that there is more than a superficial resemblance between the “learning agents” in MATSim and the (hardly understood) learning processes of real humans. If the notion of “learning” has to be used at all when interpreting Algorithm 48.1, it should be understood as “moving a MATSim model closer to its solution point”. Also see Section 97.3.5.

The remainder of this chapter phrases these statements more technically and explains their implications for the interpretation of MATSim outputs. This presentation is in parts a more technical reformulation of Chapter 47.

48.2 Relaxation as a Stochastic Process

48.2.1 Probabilistic Model Components

Algorithm 48.1 can be written more formally. Denoting the iteration index by k , the following happens in every iteration:

1. All agents choose some planned travel behavior, resulting in the travel demand D^k of the entire agent population.
2. All agents execute their travel plans, resulting in the (time-of-day dependent) network conditions C^k .
3. All agents see the resulting network conditions C^k . As a result, the information Z^k is now available to all agents.

The variables D and Z apply to the population as a whole, comprising all agents. Similarly, the variable C represents network conditions for an entire day and for the entire physical system. Given MATSim's high level of detail, one can think of D , C and Z as placeholders for arbitrarily large and complex data structures. Under MATSim's standard conditions, D corresponds to the set of selected plans, C to the collection of all events and Z to the full plans file including the scores.

Step 1 evaluates the (stochastic) travel behavior model of each agent. Technically, this comprises (i) an optional update of the plan choice set and (ii) the choice of one plan to be executed. Symbolically, this is written as

$$D^k \sim P(D \mid Z^{k-1}), \quad (48.1)$$

meaning that the travel demand of iteration k follows a probability distribution that is conditional on the information Z^{k-1} available to the agents at the end of iteration $k - 1$.

Step 2 runs the (stochastic) mobility simulation that moves all agents jointly through the network. In symbols, this becomes

$$C^k \sim P(C \mid D^k), \quad (48.2)$$

meaning that the network conditions of iteration k follow a probability distribution that is conditional on the demand D^k .

Step 3 updates the (possibly stochastic) information available to all agents using the new network conditions C^k . This is written as

$$Z^k \sim P(Z \mid C^k, Z^{k-1}). \quad (48.3)$$

That is, the new information Z^k is not only a transformation of the current network conditions C^k but may also be based on the previously available information Z^{k-1} .

The conditional distributions Equation (48.1)–(48.3) are detailed elsewhere in this book: Chapter 49 describes the plan selection mechanisms leading to $P(D \mid C^{k-1})$, Chapter 50 explains the physical processes underlying $P(C \mid D^k)$, and Chapter 3 specifies at least some of the information update logic behind $P(Z \mid C^k, Z^{k-1})$. A greater level of detail is, however, not necessary in this chapter.

48.2.2 Markov Chain Perspective

Algorithm 48.1 constitutes a discrete time stochastic process. “Discrete-time” because it evolves in stages (from iteration to iteration), stochastic because it evaluates stochastic models. Further, one iteration of this process requires only information about the previous iteration's outcome. This allows the expression of Algorithm 48.1 in terms of a “Markov chain” (Ross, 2006).

In symbols, let X^k be the Markov chain's stochastic state during stage k , and let $P(X^k = x)$ be the probability that the chain is in the concrete state x . Further, let T_y^x be the probability that the chain

enters state x in its next stage given that it is currently in state y . The transition from one stage to the next can then be expressed as follows:

$$P(X^{k+1} = x) = \sum_y P(X^k = y) \cdot T_y^x. \quad (48.4)$$

Each argument of the sum expresses the probability of the chain being in one particular state y and then entering x . The overall probability of arriving in x results from summing up these probabilities.

Markov chains tend, under certain assumptions sketched in the next section, to stabilize after sufficient iterations, in the sense that a long-term probability $\Pi(x)$ of encountering the process in state x exists. This stationary distribution satisfies

$$\Pi(x) = \sum_y \Pi(y) \cdot T_y^x, \quad (48.5)$$

which essentially results from removing the k -indices from Equation (48.4). Intuitively, removing the stage-indices k means that Equation (48.5) now applies, in the long term, for any stage k .

Given that the long-term behavior of Algorithm 48.1 shapes the predictions made with MATSim, and updated information its characterization in terms of the stationary distribution of a corresponding Markov chain is of interest. To obtain a Markov chain representation of Algorithm 48.1, one needs to specify (i) what variables in MATSim represent the states of that chain and (ii) what transition distribution underlies the MATSim simulation logic.

A state variable must provide sufficient information to simulate a process further into the future. Candidates for MATSim's state space are the demand D , the network condition C and the information Z . Of these, only the information Z qualifies as a state variable: If one knows Z^k , it is possible to draw the next day's travel demand D^{k+1} based on Equation (48.1), to insert this demand into Equation (48.2) and obtain the network conditions C^{k+1} and to finally use both C^{k+1} and Z^k to obtain an updated Z^{k+1} through Equation (48.3). This last step is what disqualifies D and C as state variables because an evaluation of Equation (48.3) is impossible without having Z in the state space.

Letting $X^k = Z^k$, the transition distribution hence needs to express how the information Z^k available to the population in iteration k carries over to the information Z^{k+1} available in iteration $k + 1$. This relationship is given by

$$T_y^x = \sum_c \sum_d P(Z^{k+1} = x \mid C^k = c, Z^k = y) P(C^k = c \mid D^k = d) P(D^k = d \mid Z^k = y). \quad (48.6)$$

Each argument of the double sum represents the probability of one particular sequence of given information y , resulting travel demand d , resulting network conditions c and updated information x . The double sum over all possible travel demand realizations d and network conditions c then accounts for the fact that there are many different such sequences through which one can start out at y and end up at x .

This completes the representation of MATSim in terms of a Markov chain. The next section illustrates practical uses of this representation.

48.3 Existence and Uniqueness of MATSim Solutions

The long-term (stationary) behavior of a Markov chain can be derived from its transition function. This also leads to useful insights for MATSim, despite of the complexity of its transition function Equation (48.6).

Two key properties are aperiodicity and irreducibility. Informally, a Markov chain is aperiodic if all of its states can be visited at irregular times; Figure (48.2) provides an example. It is irreducible if it can reach any other state from any given initial state with one or more transitions; see Figure (48.3) for an example. Aperiodicity and irreducibility are essential when it comes to long-term predictions, where (i) aperiodicity guarantees that the concrete iteration in which one evaluates the simulation does not play a role and (ii) irreducibility ensures that every possible future system state can be reached (predicted) by the simulation. If both properties are given, the Markov chain has the following properties (Ross, 2006):

1. A unique stationary distribution exists. The simulation process attains this distribution after many iterations, independently of its initial state.
2. It is feasible to compute statistics of the stationary distribution from a single simulation run, meaning that it is not necessary to run replications.

With respect to MATSim, the following holds:

- Periodicity is already broken if a nonzero probability of staying in the same state exists. This is likely to be the case in MATSim, for instance because the following sequence of events may occur by chance: (i) No agent uses plan innovation, (ii) all agents select the same plan as in the previous iteration, (iii) the mobility simulation creates identical congestion and travel time patterns as before, meaning that Z^k from Equation (48.3) remains the same as Z^{k-1} . Practically, this means that all plan scores stay unchanged.

More intuitively: Even if the system returns multiple times exactly to a state where it has been before, it is unlikely that it does so in the same number of steps.

- With plan innovation (see Sections 4.5, 4.5.3 and 47.3.2.3) switched on, irreducibility cannot be postulated:

Every time a new plan is added somewhere, the previous state space subspace where the plan was not available *cannot* be reached any more until that plan is removed; similarly, every time a plan is removed, the previous state space subspace where the plan was available cannot be

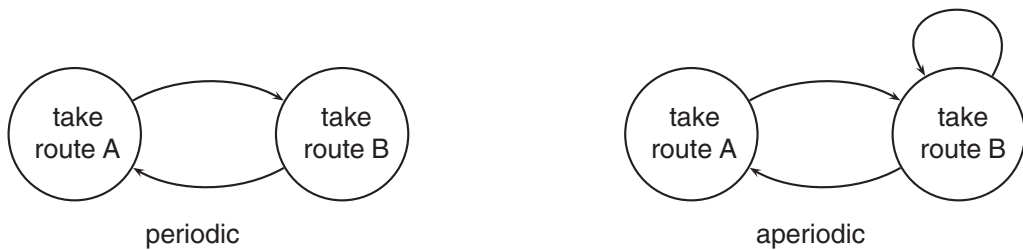


Figure 48.2: Example of (a)periodicity

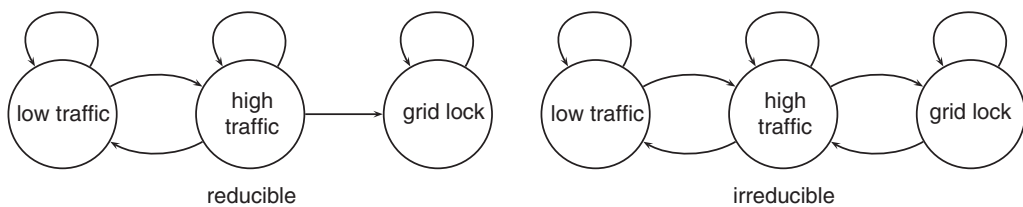


Figure 48.3: Example of (ir)reducibility

reached any more until the plan is re-created again. (Chapter 49 discusses this in greater detail and also suggests a solution for this problem.)

Even if plans creation and removal could be modeled such that irreducibility was guaranteed, the resulting process dynamics would be slow due to the state space size.

- With plan innovation switched off, MATSim in its standard configuration is likely to be irreducible. This is only “likely”, because the notion of a “standard configuration” itself is not rigorously specified here. Arguments behind this follow Cascetta (1989), who presents a related result for a much simpler, trip-based traffic simulation that only allows for route choice. Observing that travel plans are, technically, paths in a rather complicated decision network, one can then carry this result over to MATSim. See also Nagel et al. (2000) and Flötteröd et al. (2011).
- When scores are additionally forced to their expected values (Section 3.3.4), the system eventually draws agent behavior from fixed choice distributions, thus varying independently from one iteration to the next.

If config option `ChangeExpBeta` is used, some correlation is maintained between choices in subsequent iterations, even though the long-term choice distributions remain unchanged.

In summary, a mathematical framework exists allowing a rather rigorous characterization of the outcome of MATSim’s relaxation process. It turns out that MATSim, in its current form, is not necessarily a “well-behaved” stochastic process; however, casting it into this framework enables a structured approach to developing the simulation logic further. An example of how to go about this is given in Chapter 49.

48.4 Analyzing Simulation Outputs

Many of the models used in MATSim are stochastic. Examples are the discrete choice models used for plan selection or the randomized selection of the next vehicle to enter a congested downstream link in the mobility simulation. The reason for this randomness is that real mobility and transportation processes are not completely understood. The insertion of randomness represents the uncertainty remaining in the modeling.

This uncertainty may apply to both (i) model inputs, meaning that random variables are computed once before a simulation run and then kept fixed (for instance, the random generation of a synthetic population) and to (ii) processes, meaning that random variables are computed throughout the simulation (for instance, the repeated evaluation of discrete choice models). Technically, if a MATSim scenario is simulated R times with different random seeds one obtains $r = 1 \dots R$ independent simulation outputs y_r . Note that, while the raw outputs are plans and event files, the actual quantities for which y_r stands here are numerical in the majority of applications.

Given that one has used different random seeds, y_1, \dots, y_R constitute independent draws from a distribution $\Pi(Y)$. This means that if one performed a huge number of simulation runs and plotted a (possibly multidimensional) histogram of the y values, then this histogram would eventually attain the shape of $\Pi(Y)$. It is important to acknowledge that stochastic simulation outputs are a desirable consequence of stochasticity inserted elsewhere in the simulation; just as a deterministic model output is a truthful representation of its input consequences, a stochastic model output contains a truthful representation of the prediction uncertainty resulting from uncertainties in its input and its process specification.

To help intuition, one may think in the following of y_r as a large vector containing travel times on all links in all one-hour time bins as observed during the last iteration of the r th simulation run. Questions like these may then be asked:

- What travel times can one expect on average?
- What is the travel time variability?
- How probable are travel times beyond some threshold θ ?
- ...

This list can be arbitrarily continued. It turns out that most (if not all) of these questions can also be expressed symbolically. For instance:

- What travel times can one expect on average?

$$E\{Y\} = \sum_y y \cdot \Pi(Y = y) \quad (48.7)$$

This asks for the expected value of the simulation output distribution.

- What is the travel time variability?

$$\text{VAR}\{Y\} = \sum_y (y - E\{Y\})^2 \cdot \Pi(Y = y) \quad (48.8)$$

This asks for the variance (or, for multidimensional outputs, the variance-covariance matrix).

- How probable are travel times beyond some threshold θ ?

$$\Pr(Y \geq \theta) = \sum_y \mathbf{1}(y \geq \theta) \cdot \Pi(Y = y) \quad (48.9)$$

This expression merely sums up the probabilities of all simulation outputs that exceed the threshold.

- ...

This enumeration of symbols reveals a common structure. The mathematical formulation of each question can be written in the form

$$\sum_y m(y) \cdot \Pi(Y = y) \quad (48.10)$$

with different specifications of $m(y)$ (see Table 48.1).

By definition, Equation (48.10) is the expectation $E\{m(Y)\}$ given that Y is distributed according to its stationary distribution $\Pi(Y)$. Combining this with the observation that the mean over a

quantity of interest	corresponding $m(y)$
$E\{Y\}$	y
$\text{VAR}\{Y\}$	$(y - E\{Y\})^2$
$\Pr(Y \geq \theta)$	$\mathbf{1}(y \geq \theta)$
...	...

Table 48.1: Examples of m functions.

sample converges to its expectation as the number of samples grows (the Law of Large Numbers), one obtains

$$E\{m(Y)\} = \sum_y m(y) \cdot \Pi(Y = y) \quad (48.11)$$

$$= \lim_{R \rightarrow \infty} \frac{1}{R} \sum_{r=1}^R m(y_r) \quad (48.12)$$

$$\approx \frac{1}{R} \sum_{r=1}^R m(y_r) \quad \text{for a finite } R, \quad (48.13)$$

where the simulation outputs y_r , $r = 1 \dots R$, are independent draws from $\Pi(Y)$.

Now recall that initially certain questions about simulation outputs were asked. The Equation (48.11)(first row) represents exactly these questions in a formal way—and Equation (48.13) (last row) provides a simple method for computing answers to these questions. It reads as follows:

1. Define the function $m(y)$ that represents the question of interest.
2. Perform R independent simulation runs and obtain the outputs y_1, \dots, y_R .
3. Compute $m(y_r)$ for all $r = 1 \dots R$ and average these numbers.

Returning to the example questions, one thus obtains the following:

- What travel times can one expect on average?

$$E\{Y\} \approx \frac{1}{R} \sum_{r=1}^R y_r \quad (48.14)$$

Not surprisingly, this turns out to be the mean value over all simulated travel times.

- What is the travel time variability?

$$\text{VAR}\{Y\} \approx \frac{1}{R} \sum_{r=1}^R (y_r - E\{Y\})^2 \quad (48.15)$$

This is the empirical variance of the simulated travel times. (Note that in practice $E\{Y\}$ needs to be replaced by its estimator.)

- How probable are travel times beyond some threshold θ ?

$$\Pr(Y \geq \theta) \approx \frac{1}{R} \sum_{r=1}^R \mathbf{1}(y_r \geq \theta) \quad (48.16)$$

This divides the number of times the threshold was exceeded by the total number of experiments, i.e. it yields the frequency of the event of interest.

- ...

Revisiting Section 48.3, it may be possible to make these computations more efficient. If (i) there is no uncertainty in the model inputs and (ii) the simulation uses fixed choice sets, then it could be feasible to compute the above statistics by averaging over many stationary iterations of a single simulation run instead of having to run a large number of replications to convergence.

Practically, all of this is just a starting point. Important questions, such as how precise these estimates are, how many runs one needs to obtain a certain level of precision, etc. are not answered here; Ross (2006) is a good starting point for further reading.

48.5 Summary

This chapter attempted to clarify certain mechanisms underlying MATSim's iterative solution scheme. The specification of MATSim's model (components) was distinguished from MATSim's iterative solution algorithm. It was stressed that the behavioral day-to-day interpretation of MATSim is not to be taken literally; realism can only be expected from the long-term process behavior.

This long-term behavior was then related to the properties of the iteration logic using the Markov chain formalism. MATSim was phrased as such a chain, with its state space comprised of the information available for replanning. This representation was exploited to observe that the long-term distribution of MATSim is likely to exist and be unique if the plan choice sets are a priori fixed.

It further was explained that (i) there are good reasons for the stochasticity both in MATSim's inputs and outputs and that (ii) instead of avoiding stochasticity where it constitutes a truthful representation of uncertainty, one should access adequate statistical techniques to make sense of it.

CHAPTER 49

Choice Models in MATSim

Gunnar Flötteröd and Benjamin Kickhöfer

This chapter attempts to reconcile MATSim’s mechanisms of plan “mutation”, “selection” and “execution”, borrowed from evolutionary computation, with a discrete choice modeling perspective.

Discrete choice theory originates in work by Luce and Suppes (1965) and McFadden (1975); Ben-Akiva and Lerman (1985) and Train (2003) are the two standard textbooks in this area. The theory is mainly used to describe individual choices among mutually exclusive alternatives. Discrete choice models typically do not predict individual choices with complete accuracy. Luce and Suppes (1965) distinguishes between two possible interpretations of this phenomenon: (1) People choose randomly among their alternatives, rendering their behavior inherently unpredictable. (2) The choice only *appears* to be random since the model does not perfectly capture the decision process and its relevant decision variables. Both perspectives lead to the same result, the introduction of probabilistic choice models.

Let \mathcal{U}_n be the universal set of all plans that may ever be considered by agent n and let C_n denote that agent’s concrete plan choice set. The choice set independent probability that agent n selects plan i for execution can then be written as

$$P_n(i | \mathcal{U}_n) = \sum_{C_n \subset \mathcal{U}_n} P_n(i | C_n) \cdot P_n(C_n | \mathcal{U}_n), \quad (49.1)$$

explained as follows. Selecting a plan requires a plan choice set. The term $P_n(C_n | \mathcal{U}_n)$ represents the probability that this concrete choice set is C_n , which must be a subset of \mathcal{U}_n . Technically, the MATSim plan *innovation* modules draw from this distribution. The term $P_n(i | C_n)$ represents the probability that agent n selects plan i given that its concrete choice set is C_n . Technically, the MATSim plan *selection* modules draw from this distribution. The product of these terms thus

How to cite this book chapter:

Flötteröd, G and Kickhöfer, G. 2016. Choice Models in MATSim. In: Horni, A, Nagel, K and Axhausen, K W. (eds.) *The Multi-Agent Transport Simulation MATSim*, Pp. 337–346. London: Ubiquity Press. DOI: <http://dx.doi.org/10.5334/baw.49>. License: CC-BY 4.0

represents the joint probability that choice set C_n is available and that plan i is chosen from that set. The probability of selecting plan i independently of the concrete choice set then results from summing up the probabilities of selecting it in the presence of all possible choice sets $C_n \subset \mathcal{U}_n$.

It is evident in Equation (49.1) that an agent's behavior depends on both the choice model $P_n(i | C_n)$ and the way the choice set is generated through $P_n(C_n | \mathcal{U}_n)$. The following two sections will look at each step in more detail.

49.1 Evaluating Choice Models in a Simulated Environment

This section's discussion focuses on the choice distribution $P_n(i | C_n)$ for given choice sets. In MATSim, a plan is evaluated and selected based on the score as the sole property of the plan. This is only a technical specification; the scoring and selection protocols are responsible for representing adequate perceptual and behavioral mechanisms. The notions of "choice" and "selection" are subsequently used interchangeably (cf. Section 4.5.2).

The usual selection protocol of MATSim resembles a MNL choice model. Letting S_{ni} be the score of plan i of agent n , one has

$$P_n(i | C_n) = \frac{e^{\mu S_{ni}}}{\sum_{j \in C_n} e^{\mu S_{nj}}} \quad (49.2)$$

with μ controlling the preference for higher scores. It is set to one in the remainder of this section.

49.1.1 Case 1: Score is or Converges Towards a Deterministic Value

If the score of a plan was a deterministic number representing an expected value, then Equation (49.2) would constitute a plain MNL choice model with μ taking the role of a scale parameter (see, e.g., Train, 2003, p.45). Such behavior can be approximated in MATSim by the following configuration settings:

- A fixed choice set C_n is eventually obtained by setting the configuration option `fractionOfIterationsToDisableInnovation` below one, meaning that innovation (see Section 49.2) will be switched off for the remaining fraction of iterations beyond the configured value.
- Score convergence to its expectation value can be achieved by setting the configuration option `fractionOfIterationsToStartScoreMSA` below one, meaning that scores will be averaged according to MSA (Method of Successive Averages) for the remaining fraction of iterations.

49.1.2 Case 2: More General

Without the particular configuration mentioned in the previous section, things are somewhat more complicated. Assume that the attribute vector \mathbf{x}_{ni} of the alternatives in Equation (49.2) is defined through (a transformation of) the network conditions observed during the last iteration(s). Assume further that the score is a linear function of these attributes:

$$S_{ni} = \boldsymbol{\beta}^T \mathbf{x}_{ni} \quad (49.3)$$

$$= \boldsymbol{\beta}^T (\mathbb{E}\{\mathbf{x}_{ni}\} + \boldsymbol{\eta}_{ni}) \quad (49.4)$$

where β is a coefficient vector, superscript T denotes the transpose and η_{ni} is a zero mean random vector. In the general case of S_{ni} being a random variable and not just an expected value, one obtains a mixture-of-logit model with the choice distribution

$$P_n(i | C_n) = \int \frac{\exp(\beta^T E\{\mathbf{x}_{ni}\} + \beta^T \eta_{ni})}{\sum_{j \in C_n} \exp(\beta^T E\{\mathbf{x}_{nj}\} + \beta^T \eta_{nj})} p(\eta_n) d\eta_n \quad (49.5)$$

where $p(\eta_n)$ is the probability density function of $\eta_n = (\eta_{ni})_i$, i.e., the joint probability density function of the random disturbances of all alternatives of individual n (Train, 2003, Section 6). This formulation comprises most, if not all, MATSim configurations currently used. It represents the ExpBetaPlanSelector and the equivalent ExpBetaPlanChanger. It also comprises the BestPlanSelector, because that is equivalent to the ExpBetaPlanSelector with a very large (infinite) μ . Arbitrary score averaging schemes are also included; this only leads to different instances of $p(\eta_n)$.

Mainstream applications of mixture-of-logit models attempt to combine the tractability of closed-form logit models with the flexibility of simulating arbitrary $p(\eta_n)$ distributions. The distribution of η_n is often as simple as a multivariate normal because this already allows for the introduction of rich correlation structures into the underlying random utilities. In MATSim, however, the simulated error term η_n is extremely complicated. Revisiting Equation (49.4), it defines the variability of the scores resulting from the fact that the simulated network conditions are stochastic. The distribution from which these network conditions are drawn is defined implicitly through the mobility simulation. It is not available in closed form; one can only draw from it.

Additional complexity results from the simulated network conditions being, in turn, the consequence of simulated travel behavior that is again defined through Equation (49.5). Just as a representation of the mutual demand/supply dependency is essential in transport planning, the circular definition of the η_n terms adds realism to MATSim:

1. Assume one could somehow make the simulated network conditions more realistic. The result would be a more realistic distribution $p(\eta_n)$ of the simulated error terms.
2. All else equal, increasing the realism of $p(\eta_n)$ in Equation (49.5) would also increase the realism of the resulting choice distribution.
3. This, in turn, would lead to the selection of more realistic travel plans, meaning that their execution would result in even more realistic network conditions.

However, this positive feedback only applies to the extent to which the error terms in the behavioral model are indeed mobility simulation outputs. Simulated travel time (variability) is such a case. Unobserved preferences of the decision maker, however, are not an output of the mobility simulation and hence need to be differently captured.

It is by no means obvious how the randomness of the simulated network conditions should enter η_n . The notion of “learning” again enters the picture, cf. Chapter 48. However, if the simulation iterations really represented simulated days then a real human learning model would be needed to combine a sequence of past network conditions into an instantaneous η_n realization. Without a sound instance of such a learning model, a learning-based interpretation of Equation (49.5) cannot be given.

Another perspective on this problem is possible, continuing the arguments of Chapter 48. It is stated there that the purpose of MATSim’s iterative mechanism is merely to attain a realistic stationary distribution. If so, then the sole purpose of the simulated η_n s is to yield a realistic

stationary choice distribution. To illustrate this perspective, consider the following moving-average score updating rule:

$$\bar{S}_{ni}^{k+1} = \begin{cases} \alpha S_{ni}^k + (1 - \alpha) \bar{S}_{ni}^k & \text{if } n \text{ chose plan } i \\ \bar{S}_{ni}^k & \text{otherwise} \end{cases} \quad (49.6)$$

where \bar{S}^k is the filtered score of iteration k and S^k is the concrete score observed in that iteration. The learning rate α controls how strongly the filtered score is smoothed out, thus controlling the variability of η_n . MATSim enables this mechanism through the `learningRate` parameter.

Assuming – for simplicity – that the unfiltered stationary score S^∞ fluctuates in stationary conditions independently from iteration to iteration around its expected value, one can derive the following (as demonstrated in this chapter's appendix):

$$E\{\bar{S}_{ni}^\infty\} = E\{S_{ni}^\infty\} \quad (49.7)$$

$$\text{VAR}\{\bar{S}_{ni}^\infty\} = \frac{\alpha}{2 - \alpha} \text{VAR}\{S_{ni}^\infty\}. \quad (49.8)$$

This means that the filtered score is unbiased with respect to the underlying score process and that its variance is in the interval from zero to the variance of the unfiltered score, depending on the chosen α . There is no need to justify this through a learning process. One has merely constructed a parametrization of the distribution $p(\eta_n)$. In the resulting mixture model Equation (49.5), α should be estimated from real data, just like any other model parameter. Even though this apparently has not yet been attempted, techniques necessary for such an endeavor are, in principle, available (Gourieroux et al., 1993).

49.1.3 Expected Maximum Utility

The expected maximum utility of Equation (49.5) is relevant to the microeconomic interpretation of MATSim outputs. A recipe for its computation is described next. Let

$$U_i = V_i + \eta_i + \varepsilon_i \quad (49.9)$$

using the shortcuts $V_i = \beta^T E\{\mathbf{x}_{ni}\}$, $\eta_i = \beta^T \eta_{ni}$, letting ε_i be the Gumbel error assumed by the multinomial logit model and dropping the n index for brevity. Following this notation, Equation (49.4) is rewritten as

$$S_i = V_i + \eta_i. \quad (49.10)$$

One needs to distinguish between the score of a plan when it is selected and its updated score after it has been executed. To start, it is assumed that the agent receives an expected maximum utility depending on the scores at the time of plan selection, not after plan execution. The expected maximum utility of Equation (49.5) could then be expressed as follows:

$$E\left\{\max_{i \in C_n} U_i\right\} = E\left\{\max_{i \in C_n} V_i + \varepsilon_i + \eta_i\right\} \quad (49.11)$$

$$= E_\eta \left\{ E_\varepsilon \left\{ \max_{i \in C_n} V_i + \varepsilon_i + \eta_i \mid \eta \right\} \right\} \quad (49.12)$$

$$= E_\eta \left\{ \ln \sum_{i \in C_n} e^{V_i + \eta_i} \right\}. \quad (49.13)$$

where the law of total expectation is used and E_ε and E_η represent expectations with respect to ε and η , respectively. The remaining argument of the expectation is the expected maximum utility of a multinomial logit choice model given the systematic utilities $V_i + \eta_i$. This expression can be numerically approximated by averaging over many realizations of η_i (i.e. over simulation iterations):

$$E_\eta \left\{ \ln \sum_{i \in C_n} e^{V_i + \eta_i} \right\} \approx \frac{1}{R} \sum_{r=1}^R \ln \sum_{i \in C_n} e^{V_i + \eta_i^r} \quad (49.14)$$

where η_i^r is the realization of η_i in iteration r . This expression holds regardless of the functional form of the mobsim-generated mixture distribution.

Now, one needs to account for the fact that agents can only evaluate *past* information when making a choice leads to a *future* score payoff. Recalling that score variability is represented by the η_i variables in Equation (49.5),

$$\eta_i = \hat{\eta}_i + \gamma_i \quad (49.15)$$

is written with η_i contributing to the score actually received Equation (49.10), $\hat{\eta}_i$ being the agent's prediction of that and γ_i being a random variable capturing the difference between the two.

To express the expected maximum *experienced* utility, one hence needs to add (an estimator of) the expectation of γ_i to Equation (49.14). Using Equation (49.15) and Equation (49.10), one obtains

$$\gamma = \eta_i - \hat{\eta}_i \quad (49.16)$$

$$= (\eta_i + V_i) - (\hat{\eta}_i + V_i) \quad (49.17)$$

$$= S_i - \hat{S}_i \quad (49.18)$$

where \hat{S}_i can be interpreted as the agent's prediction of the selected alternative i 's score. The expectation of this quantity can again be approximated by averaging, resulting in the following estimator of the expected maximum experienced utility, with $i(r)$ indicating the alternative that was selected in iteration r :

$$E \left\{ \max_{i \in C_n} U_i^{\text{experienced}} \right\} \approx \frac{1}{R} \sum_{r=1}^R \ln \sum_{i \in C_n} e^{\hat{S}_i^r} + \frac{1}{R} \sum_{r=1}^R (S_{i(r)}^r - \hat{S}_{i(r)}^r). \quad (49.19)$$

The second sum of this expression estimates a “cost of uncertainty”; the less predictable the network conditions (and thus the selected plan's future score), the worse off an agent is on average. The usefulness of this expression depends on the simulation's ability to create realistic network condition variability, for instance along the lines of the last paragraphs of Section 49.1.2. Section 51.2.5.5 discusses this a bit further.

49.2 Evolution of Choice Sets in a Simulated Environment

49.2.1 Overview

The choice set of agents can in principle be computed a priori and then held fixed during a MATSim simulation run. However, the pre-computation would have to be done for every relevant system state (e.g., before and after a policy change). Alternatively, MATSim can be used to generate agents' choice sets within the iterative loop (Section 1.2).

As Equation (49.1) shows, the generation of the choice sets affects the simulated choices. The simplest illustration of this mechanism is that alternatives that never appear in the choice set cannot be chosen. Similarly, including certain alternatives with a low (high) probability in the choice set decreases (increases) their probability of being chosen, given that the choice model is not changed. When a policy study's synthetic choice sets are very different from the alternatives considered in the real world, it is unlikely that the simulation will display correct aggregated quantities or useful sensitivities for policy measures.

These types of biases are well-known in the discrete choice community, even though the focus is there, arguably, more on estimation than simulation. The problem is particularly acute in route choice modeling because the combinatorial size of the universal route choice set prohibits its enumeration. Drawing further from the discrete choice literature (specifically Frejinger and Bierlaire, 2010), different interpretations can be given to “plan mutation” and “plan innovation” in MATSim.

An interpretation of mutation and innovation as *perceptual models* of travel plan choice set formation is hindered by the need to validate them against real and unobservable choice sets. Alternatively, one may assume that travelers consider the universal choice set and that the choice of unfeasible alternatives is impeded by correspondingly low utility values. In this setting, mutation and innovation constitute *sampling techniques* serving the *computational* purpose of reducing the universal choice set to a small, representative subset. However, one still faces the problem from above that the concrete sampling protocol has a concrete effect on the simulated behavior. The cure when *estimating* choice models is to correct for the sampling based on known sampling probabilities (e.g. Ben-Akiva and Lerman, 1985, Chapter 9), even though these probabilities can be rather difficult to obtain (Flötteröd and Bierlaire, 2013; Frejinger et al., 2009a). The problem appears to be less explored when it comes to *simulation*.

MATSim's currently implemented mutation and innovation procedures constitute concrete, yet heuristic, approaches to the choice set generation problem, aiming at valid predictions at the system level. Possible biases induced by these procedures can, however, be difficult to quantify. For example, the current MATSim implementation might, under certain conditions, yield incomplete choice sets and correlated alternatives (also see Chapter 51). To mitigate the effect of strong correlations between alternatives within the choice set, so-called diversity increasing re-planning modules have been tested (see, e.g., Nagel et al., 2014). In the same context, Grether (2014, Chapter 6) and Neumann et al. (2013) have tested path size logit approaches (see, e.g., Daganzo and Sheffi, 1977; Frejinger and Bierlaire, 2007) to maintain diversity in the choice set by penalizing similar alternatives. Still, these approaches are—as of now—ad-hoc solutions, with little theoretical foundation in the simulation context.

It thus seems worthwhile to revisit the plan choice set generation problem from a statistical perspective. The goal of the following presentation is more to establish a corresponding mindset than deliver a complete solution.

49.2.2 Towards Unbiased Choice Set Generation

To make the simulated long-term (stationary) plan choice independent of the plan choice set generation, one may require the following stationary choice distribution:

$$P_n(i \mid \mathcal{U}_n) = \frac{e^{\mu S_{ni}}}{\sum_{j \in \mathcal{U}_n} e^{\mu S_{nj}}}, \quad (49.20)$$

meaning that plans are selected from the universal choice set \mathcal{U}_n .

Denoting by $P(C_n \rightarrow C'_n)$ the probability that plan mutation/innovation turns the choice set C_n into C'_n , it is possible to enforce the long-term choice distribution Equation (49.20) through an application of the MH (Metropolis-Hastings) algorithm (Hastings, 1970, see also Flötteröd and Bierlaire (2013) for a related approach to a similar problem).

The MH algorithm specifies the transition distribution of a Markov chain so that a desired stationary distribution of that chain is reached. Given that Chapter 48 has established a formulation of MATSim as such a chain, the MH machinery can hence be inserted into the MATSim iterations. A simplification made in the following is that the choice distribution of agent n is considered independent of all other agents.

To make this concrete, let the state space of the algorithm be the tuple $(C_n, i \in C_n)$ consisting of choice set and resulting choice. During each (MATSim) iteration, one first draws a new choice set C'_n , then draws a new choice $i' \in C'_n$ according to the usual model (49.2) and finally accepts the new state (C'_n, i') with probability

$$\phi[(C_n, i), (C'_n, i')] = \min \left\{ \frac{P_n(i' | \mathcal{U}_n)}{P_n(i | \mathcal{U}_n)} \cdot \frac{P(C'_n \rightarrow C_n)P_n(i | C_n)}{P(C_n \rightarrow C'_n)P_n(i' | C'_n)}, 1 \right\} \quad (49.21)$$

and rejects it otherwise (meaning that the original choice set C_n and choice $i \in C_n$ are maintained).¹ Intuitively, the first fraction introduces a preference for states comprising a more probable choice and the second fraction corrects for the way transitions between states are proposed.

Assume that the plan innovation yields exactly one new plan i_{in} through a against the last iteration. Let the corresponding plan innovation distribution be approximated by $e^{\mu_{\text{inno}} S_{ni}} / \sum_{j \in \mathcal{U}_n} e^{\mu_{\text{inno}} S_{nj}}$ with a very large μ_{inno} . Assume further that i_{in} replaces exactly one uniformly selected plan i_{out} , which implies that the choice set size J is constant and exclude for simplicity the case that the best response innovation reconstructs the removed plan exactly. This leads to

$$P(C_n \rightarrow C'_n) = \frac{1}{J} \cdot \frac{e^{\mu_{\text{inno}} S_{ni_{\text{in}}}}}{\sum_{j \in \mathcal{U}_n} e^{\mu_{\text{inno}} S_{nj}}} \quad (49.22)$$

$$P(C'_n \rightarrow C_n) = \frac{1}{J} \cdot \frac{e^{\mu_{\text{inno}} S_{ni_{\text{out}}}}}{\sum_{j \in \mathcal{U}_n} e^{\mu_{\text{inno}} S_{nj}}} \quad (49.23)$$

Inserting this as well as Equation (49.2) and Equation (49.20) into Equation (49.21), one obtains

$$\phi[(C_n, i), (C'_n, i')] = \min \left\{ \frac{e^{\mu_{\text{inno}} S_{ni_{\text{out}}}} \frac{e^{\mu_{\text{inno}} S_{ni}}}{\sum_{j \in C_n} e^{\mu_{\text{inno}} S_{nj}}}}{e^{\mu_{\text{inno}} S_{ni'}} \frac{e^{\mu_{\text{inno}} S_{ni_{\text{in}}}}}{\sum_{j \in C'_n} e^{\mu_{\text{inno}} S_{nj}}}}, 1 \right\} \quad (49.24)$$

$$= \min \left\{ e^{\mu_{\text{inno}} (S_{ni_{\text{out}}} - S_{ni_{\text{in}}})} \cdot \frac{\sum_{j \in C'_n} e^{\mu_{\text{inno}} S_{nj}}}{\sum_{j \in C_n} e^{\mu_{\text{inno}} S_{nj}}}, 1 \right\} \quad (49.25)$$

$$\mu_{\text{inno}} \xrightarrow{=} \infty \quad \begin{cases} 1 & \text{if } S_{ni_{\text{out}}} \geq S_{ni_{\text{in}}} \\ 0 & \text{otherwise.} \end{cases} \quad (49.26)$$

¹ The acceptance probability $\phi(X \rightarrow X')$ in MH sampling is calculated as

$$\min \left(\frac{w(X') \cdot p_{\text{propose}}(X' \rightarrow X)}{w(X) \cdot p_{\text{propose}}(X \rightarrow X')}, 1 \right),$$

where $p_{\text{propose}}(\cdot \rightarrow \cdot)$ is the probability that a certain transition is proposed, and $w(X)$, $w(X')$ are the relative weights of the respective states. It is important to note that w does not have to be normalized; it is sufficient if $w(X)/w(X') = p(X)/p(X')$. $P(C \rightarrow C')P(i'|C')$ is the probability that the choice set transitions from C to C' and that i' is selected from the resulting choice set.

Some care is needed when evaluating this expression because it assumes $S_{ni_{in}}$ and $S_{ni_{out}}$ to be independent random variables, whereas $S_{ni_{in}}$ is (due to the best response) always maximal among all alternatives given the most recent iteration. One should thus evaluate this expression by computing either score from the network conditions of a different, randomly selected stationary iteration.

This would allow the selection of plans according to (49.20) from an unconstrained choice set, even though one enumerates only a small subset of the full choice set, which is updated through a computationally efficient best-response mechanism.

In summary, one does the following for each agent in each iteration:

1. Randomly select a given plan for removal and compute a new best-response plan against the last iteration.
2. Is the new plan better than the one selected for removal, based on network conditions from two randomly selected stationary iterations?
 - Yes: Keep the previously selected plan and the previous choice set.
 - No: Remove the randomly selected plan from the choice set, add the newly generated plan and select a new plan from the new choice set.

This (at first glance perhaps counter-intuitive) logic can be explained as follows: Best-response creates new plans that are by chance better than any other plan in a given iteration. Best-response is thus corrected for by accepting the new plan only if it is by chance worse than a randomly selected alternative plan, with both plans being evaluated in randomly selected stationary iterations.

Note that the accuracy of this approach depends on the ability of the best-response plan innovation to create sufficiently variable plans, in the sense that the plan choice set innovation process is irreducible (Ross, 2006, see also Section 48.3 for an intuitive definition of irreducibility).

49.3 Summary

This chapter attempted to phrase MATSim's mechanisms of plan scoring, innovation, mutation and selection in the more mainstream terminology of discrete choice modeling. The implications of evaluating stochastic scores when selecting a plan were explained. The chapter also addressed how simulated choices depend on the way the underlying plan choice sets are generated, and different ways to address this problem were described.

The chapter clearly brought up more issues than it resolved. The take-away message, if any, is probably that even though MATSim agent behavior is roughly based on discrete choice modeling, one needs to be careful when assuming full consistency with a particular discrete choice model.

Appendix: Derivation of Filtered Score Statistics

Writing out the expectation:

$$E\{\bar{S}_{ni}^{k+1}\} = P_n(i)E\{\alpha S_{ni}^k + (1 - \alpha)\bar{S}_{ni}^k\} + (1 - P_n(i))E\{\bar{S}_{ni}^k\} \quad (49.27)$$

$$\Leftrightarrow E\{\bar{S}_{ni}^{k+1}\} - E\{\bar{S}_{ni}^k\} = \alpha P_n(i)(E\{S_{ni}^k\} - E\{\bar{S}_{ni}^k\}). \quad (49.28)$$

From $\lim_{k \rightarrow \infty} E\{\bar{S}_{ni}^{k+1}\} - E\{\bar{S}_{ni}^k\} = 0$ then follows

$$\lim_{k \rightarrow \infty} (E\{S_{ni}^k\} - E\{\bar{S}_{ni}^k\}) = 0. \quad (49.29)$$

Proceeding in a similar way for the second moment:

$$\begin{aligned} E\{(\bar{S}_{ni}^{k+1})^2\} &= P_n(i)E\{(\alpha S_{ni}^k + (1-\alpha)\bar{S}_{ni}^k)^2\} \\ &\quad + (1-P_n(i))E\{(\bar{S}_{ni}^k)^2\} \end{aligned} \quad (49.30)$$

$$\begin{aligned} \Leftrightarrow \lim_{k \rightarrow \infty} E\{(\bar{S}_{ni}^{k+1})^2\} - E\{(\bar{S}_{ni}^k)^2\} &= P_n(i)\alpha \left[\alpha E\{(S_{ni}^k)^2\} \right. \\ &\quad \left. + 2(1-\alpha)E\{S_{ni}^k\}^2 - (2-\alpha)E\{(\bar{S}_{ni}^k)^2\} \right] \end{aligned} \quad (49.31)$$

From $\lim_{k \rightarrow \infty} E\{(\bar{S}_{ni}^{k+1})^2\} - E\{(\bar{S}_{ni}^k)^2\} = 0$ then follows

$$\lim_{k \rightarrow \infty} E\{(\bar{S}_{ni}^k)^2\} = \frac{\alpha}{2-\alpha} E\{(S_{ni}^k)^2\} - \frac{2-2\alpha}{2-\alpha} E\{S_{ni}^k\}^2. \quad (49.32)$$

The limiting variance then results from inserting of Equation (49.29) and Equation (49.32) into

$$\lim_{k \rightarrow \infty} \text{VAR}\{\bar{S}_{ni}^k\} = \lim_{k \rightarrow \infty} \left[E\{(\bar{S}_{ni}^k)^2\} - E\{\bar{S}_{ni}^k\}^2 \right] \quad (49.33)$$

$$= \frac{\alpha}{2-\alpha} \text{VAR}\{(S_{ni}^k)^2\}. \quad (49.34)$$

Queueing Representation of Kinematic Waves

Gunnar Flötteröd

50.1 Introduction

MATSim comes with a number of mobsims (cf. Sections 4.3, 43.1); the most important are the so-called QSim and JDEQSim. These differ from the implementation perspective (time-stepping vs. event-based, degree of parallelism), but all are (at least approximate) solvers of the same underlying traffic flow model. The purpose of this chapter is to relate MATSim's mobsims to the existing traffic flow theory. There are other simulation packages rooted in the same underlying modeling concepts (Tian et al., 2007; Zhou and Taylor, 2014).

The flow-density relationship (also called FD (Fundamental Diagram)) shown in Figure 50.1 is at the heart of MATSim's traffic flow model. Given a long, homogeneous road, it predicts average flow q (in vehicles per time unit) through any cross-section of that road, given an average vehicle density ϱ (in vehicles per length unit) on that road.

The FD is defined as the minimum of a sending function $S(\varrho)$ (solid) and a receiving function $R(\varrho)$ (dashed), resulting overall in a triangular curve parametrized by free flow speed v , maximum density $\hat{\varrho}$ and backward wave speed w . The maximum velocity is an observable parameter that can be set in the network file (`freespeed` attribute of the `link` element). The maximum density equals one over the length of a vehicle (`effectivecellsize` attribute of the `links` element) for a single-lane link and needs to be multiplied with the number of lanes (`permlanes` attribute of the `link` element), otherwise. The backward wave speed turns out to be the (negative of the) ratio of vehicle length to the safety time gap adopted by drivers in congested conditions. This parameter is fairly constant; a vehicle length of 7.5 meters and a time gap of 2 seconds leads to a value of (minus) 13.5 kilometers per hour. The backward wave speed can be set in the JDEQSim through the `gapTravelSpeed` parameter; it cannot currently be set in the QSim.

The considered FD alone applies only in stationary conditions, where it predicts that (i) flow increases linearly with density at low densities (i.e., in uncongested conditions); (ii) flow decreases

How to cite this book chapter:

Flötteröd, G. 2016. Queueing Representation of Kinematic Waves. In: Horni, A, Nagel, K and Axhausen, K W. (eds.) *The Multi-Agent Transport Simulation MATSim*, Pp. 347–352. London: Ubiquity Press. DOI: <http://dx.doi.org/10.5334/baw.50>. License: CC-BY 4.0

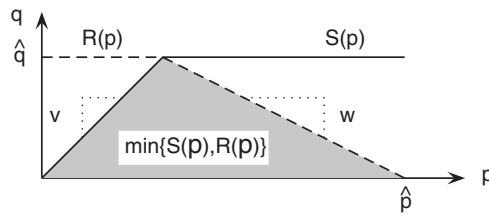


Figure 50.1: Fundamental diagram.

linearly with density at high densities (i.e., in congested conditions); and (iii) in between, it attains a maximal value constituting the flow capacity

$$\hat{q} = \frac{vw\hat{q}}{v+w} \quad (50.1)$$

of the link. This parameter represents the maximum throughput of the link in the absence of any other flow constraint (such as downstream traffic lights or other bottlenecks, which are discussed further below).

A realistic representation of non-stationary traffic flow (where density and flow change over space and time) is possible by inserting the FD into a continuity equation (which intuitively models vehicle conservation, in the sense that vehicles cannot vanish or spontaneously appear on a road segment without on- and off-ramps). This leads to the KWM (Kinematic Wave Model) of traffic flow (Lighthill and Witham, 1955; Richards, 1956), where the sending and receiving function receive an intuitive interpretation: The instantaneous flow across any interface, possibly with different densities prevailing and FDs applying up- and downstream of that interface, is defined by (i) inserting the density upstream of the interface into the upstream sending function, (ii) inserting the density downstream of that interface into the downstream receiving function and (iii) taking the minimum of these two quantities (Daganzo, 1994; Lebacque, 1996). Intuitively: The flow is limited by what can be sent from upstream and what can be received downstream, but otherwise it is maximized.

The remainder of this chapter expresses MATSim’s link model (Section 50.2) and its node model (Section 50.3) in terms of the sending and receiving function framework of the KWM. Some technical detail is omitted from the presentation for the sake of readability; pointers to the literature are provided.

50.2 Link Model

To compute flows entering and leaving a link, one needs to know how much flow can maximally enter the link and how much flow can maximally leave the link. Both constraints depend on the internal (congestion) state of the link. In symbols, one is interested in the instantaneous receiving flow rate R of the link’s upstream end and the instantaneous sending flow rate S of the link’s downstream end. Multiplying these rates by the duration δ of a simulation time step then yields the maximum number of vehicles that can enter or leave the link during a time step.

MATSim also needs to compute these quantities; how it does so is rooted in Newell’s “simplified theory of kinematic waves” (Newell, 1993), which provides a tracktable recipe for computing flow and density anywhere in a link, given that one keeps track only of the flows at the link’s up- and downstream interface. In the continuum model (i.e., one that allows for real-valued flows and densities at real-valued locations and times) specified by Newell (1993), the cumulative in- and

outflow of a link are defined as

$$N^{\text{in}}(t) = \int_0^t q^{\text{in}}(z) dz \quad (50.2)$$

$$N^{\text{out}}(t) = \int_0^t q^{\text{out}}(z) dz \quad (50.3)$$

where t denotes time, q^{in} and q^{out} are the instantaneous in- and outflow rates (in vehicles per time unit) of the link and an initially (at $t = 0$) empty link is assumed. From MATSim's vehicle-discrete perspective, cumulative inflow (outflow) at a given point in time hence represents the total number of vehicles having entered (left) the link up to that point in time.

Yperman et al. (2006); Yperman (2007) observe that if Newell's theory allows computation of instantaneous densities anywhere in a link, then it also allows computation of densities at the up- and downstream ends of that link. Inserting these densities in the link's sending and receiving function then allows expressing the sending and receiving flows as functions of time-shifted cumulative in- and outflows only, with the time-shifts specified according to the original Newell (1993) formula:

$$R(t) = \min \{ \hat{q}L - [N^{\text{in}}(t) - N^{\text{out}}(t + \delta - L/|w|)], \hat{q}\delta \} \quad (50.4)$$

$$S(t) = \min \{ [N^{\text{in}}(t + \delta - L/v) - N^{\text{out}}(t)], \hat{q}\delta \} \quad (50.5)$$

where L is the link length and δ is the (small) discrete time step length. Yperman (2007) provides some intuition for this rather formal specification.

The connection to MATSim can now be made explicit by labeling the two bracketed terms in Equation (50.4) and Equation (50.5) as "upstream queue" (UQ) and "downstream queue" (DQ) (Osorio et al., 2011; Osorio and Flötteröd, published online in *Articles in Advance*):

$$\text{UQ}(t) = N^{\text{in}}(t) - N^{\text{out}}(t + \delta - L/|w|) \quad (50.6)$$

$$\text{DQ}(t) = N^{\text{in}}(0, t + \delta - L/v) - N^{\text{out}}(t). \quad (50.7)$$

These expressions can be given a recursive meaning. Evaluating $\text{UQ}(t) - \text{UQ}(t - \delta)$ yields $[N^{\text{in}}(t) - N^{\text{in}}(t - \delta)] - [N^{\text{out}}(t + \delta - L/|w|) - N^{\text{out}}(t - L/|w|)]$, which under the assumption that flow rates are held constant throughout a simulation time step simplifies into $\delta[q^{\text{in}}(t - \delta) - q^{\text{out}}(t - L/|w|)]$. From this (and symmetric operations for DQ), one obtains

$$\text{UQ}(t) = \text{UQ}(t - \delta) + \delta[q^{\text{in}}(t - \delta) - q^{\text{out}}(t - L/|w|)] \quad (50.8)$$

$$\text{DQ}(t) = \text{DQ}(t - \delta) + \delta[q^{\text{in}}(t - L/v) - q^{\text{out}}(t - \delta)]. \quad (50.9)$$

These recursive definitions turn out to be the continuum version of how the JDEQSim updates its link model: In every time step, all vehicles that have just left the link are taken out of the DQ and all vehicles that have entered the link L/v time units ago (corresponding to free-flow travel time) are inserted into the DQ. Similarly, all vehicles that have just entered the link are put into the UQ and all vehicles that have left the link $L/|w|$ time units ago are only now taken out of the UQ. Further, inserting (50.6) and (50.7) into (50.4) and (50.5) yields

$$R(t) = \min \{ \hat{q}L - \text{UQ}(t), \hat{q}\delta \} \quad (50.10)$$

$$S(t) = \min \{ \text{DQ}(t), \hat{q}\delta \}, \quad (50.11)$$

which again corresponds to how JDEQSim evaluates the boundary conditions of a link: The amount of flow allowed to enter the link is limited by the space in its UQ and the amount of flow allowed to leave the link is limited by the number of vehicles in its DQ.

A mobsim that implements the rules Equation (50.8), Equation (50.9), Equation (50.10) and Equation (50.11) implements a KWM-consistent link model. This is almost the case for the JD-EQSim, which, in its implementation as of December 2014, exhibits the sole inconsistency of not limiting the link's inflow to its flow capacity. The QSim turns out to be a particular instance of the same model where backward wave speed is set to $|w| = L/\delta$. Inserting this into Equation (50.8) leads to

$$UQ(t) = UQ(t - \delta) + \delta[q^{\text{in}}(t - \delta) - q^{\text{out}}(t - \delta)], \quad (50.12)$$

which represents the total number of vehicles in the entire link. This corresponds to QSim behavior, where inflow to a link is limited only by the available space in the link as a whole. Letting $|w| = L/\delta$ means that the QSim behaves like a KWM with an extremely high backward wave speed, which physically means that a queue on the link does not dissolve from its downstream end but moves "en block" over the link.

50.3 Node Model

All mobsims in MATSim implement the same node model. Surprisingly, this node model can be traced back at least to (Cetin et al., 2003, under the name of "fair intersections"), while the literature establishing its consistency with the KWM is only a few years old (Tampere et al., 2011; Flötteröd and Rohde, 2011; Corthout et al., 2012).

Nodes in MATSim have no spatial dimension; they merely connect up- and downstream links. Tampere et al. (2011) specify a set of requirements for a (continuum) node model to be consistent with the KWM. They require that the flow through the node shall be maximized subject to the following constraints:

1. Flows are non-negative and conserved within the node. This means that vehicles cannot drive backwards and they must neither disappear nor appear within the node.
2. Flow ratios comply with exogenously specified turning fractions. For instance, if it is specified that 20 % of the outflow of link i shall turn into link j , then the amount of flow that actually advances from link i into link j shall indeed be 20 % of the flow that actually leaves link i .
3. Sending flows of upstream links and receiving flows of downstream links are not exceeded. This is explained in Section 50.2.
4. The invariance principle of Lebacque and Khoshyaran (2005) is satisfied. The most important intuitive implication of this principle is that the advancement of a queuing vehicle is not affected by the vehicles behind it.
5. A "supply constraint interaction rule" is satisfied. It defines how the limited receiving flow of a downstream link is shared by competing upstream links: in practical terms, a right-of-way specification.

Flötteröd and Rohde (2011) specify an "incremental node model" that satisfies these requirements and also provide an intuitive, computationally efficient solution algorithm. In each simulation time step, this algorithm incrementally (hence the name) moves flow from upstream links into downstream links. It does so such that all the previously enumerated constraints are satisfied anytime during the transfer, terminating only once no more flow can be moved. Thus, the ultimately moved flows also comply with all constraints and are maximal.

Now consider the code documentation of MATSim's `queuesim.QueueNode.moveNode` (as of December 2014):

```
Moves vehicles from the inlinks' buffer to the outlinks where
possible. The inLinks are randomly chosen, and for each link all
vehicles in the buffer are moved to their desired outLink as long as
there is space. If the front vehicle in a buffer cannot move across
the node because there is no free space on its destination link,
the work on this inLink is finished and the next inLink's buffer is
handled.
```

This is an informal description of how the incremental node model of Flötteröd and Rohde (2011) works, given that one adopts the conventions that the sending flow of a link is stored in its “buffer” and that the receiving flow of a link is labeled here as free space in (the upstream queue of) that link. A more detailed inspection of the underlying implementation reveals no inconsistencies with incremental node model specification.

There are two aspects of the MATSim node model that are not reflected by the above code comment.

- The sending flow of an upstream link may be limited by an outflow capacity below the flow capacity Equation (50.1) of that link; for instance, to approximate a capacity reduction resulting from a downstream traffic light. This is still consistent with the framework described above.
- The selection probability of “inLinks” is proportional to their flow capacity, meaning that links with higher capacity send, on average, more flow. This is again consistent with Flötteröd and Rohde (2011) and constitutes a concrete “supply constraint interaction rule”, as required by Tampere et al. (2011).

The relative simplicity of MATSim's intersection logic may be refined in many ways. For instance, turning pockets may be added and conflicts within intersections may be modeled (cf. Chapter 12). However, some caution is needed when implementing such extensions. The present node model is, due to its simplicity, guaranteed to yield unique node flows. This property needs to be revisited when implementing more complicated specifications (Corthout et al., 2012).

50.4 Summary

This chapter demonstrated that MATSim's mobility simulation is already very close to implementing a particle-discretized instance of the KWM. For full consistency, one needs to (i) use the JDEQSim (or to implement a realistic backward wave speed in the QSim) and to (ii) limit the inflow of a link by its flow capacity (which corresponds to the maximum of its triangular FD).

CHAPTER 51

Microeconomic Interpretation of MATSim for Benefit-Cost Analysis

Benjamin Kickhöfer and Kai Nagel

This chapter explains how MATSim's agent-based framework can be interpreted from a micro-economic perspective and how it can be used for the economic evaluation of transport policies, e.g., for BCA (Benefit-Cost Analysis). The text of this chapter is partly taken from Kickhöfer (2014, Section 2.3).

Typically, the process of economic policy evaluation consists of three steps: First, forecasting changes in the system by modeling users' reactions to a policy (Section 51.1). Second, assigning some (potentially monetary) valuation to these changes (Section 51.2). And third, applying an appropriate aggregation rule (Section 51.3). As will be shown in the next sections, these steps are neither completely independent nor completely dependent on each other.

51.1 Revisiting MATSim's Behavioral Simulation

Estimating policy intervention benefits relies on a sound descriptive model able to predict individuals' related behavioral changes. As explained in Section 1.2, agents in MATSim optimize their mobility behavior over several iterations by reacting to the behavior of other agents. Even if one assumes homogeneous individual preferences in the behavioral parameters of their utility functions (see Section 3.4), activity locations and activity patterns of agents typically differ, meaning that the simulation deals with heterogeneous decision makers. It thus seems reasonable to interpret the simulation from a discrete choice modeling perspective (see Chapter 49). Another attractive reason to use this interpretation lies in the well-established approaches to estimate user benefits and system welfare changes in discrete choice models.

How to cite this book chapter:

Kickhöfer, B and Nagel, K. 2016. Microeconomic Interpretation of MATSim for Benefit-Cost Analysis. In: Horni, A, Nagel, K and Axhausen, K W. (eds.) *The Multi-Agent Transport Simulation MATSim*, Pp. 353–364. London: Ubiquity Press. DOI: <http://dx.doi.org/10.5334/baw.51>. License: CC-BY 4.0

As shown by Nagel and Flötteröd (2012, also see Chapter 47 and Section 49.1.1), the MATSim choice model is equivalent to a standard MNL model under the following two conditions: first, valid choice sets have been found for all individuals; second, the score of each plan has converged to its expectation value (self-consistent state). An approximation of this can be reached by switching innovation off (Section 4.5.3) and forcing scores to convergence (Section 3.3.4, also see Section 49.1.1). Still, the following methodological issues remain:

1. **Choice set incomplete:** The maximum number of plans J in each agent's choice set is limited by memory constraints; the choice set for decision making is, hence, unlikely to be complete.
2. **Plans correlation from innovation:** Plans might be correlated. This is very likely if they are modified or replaced by best-response re-planning modules (e.g., the route choice module), since they always have a tendency to generate the same answer. However, random mutations, in general, also tend to result in correlated plans, since the concept of a mutation implies only a small move away from the parent. This violates the required IIA (Independence from Irrelevant Alternatives) property of the choice set necessary for a MNL model.
3. **Plans correlation from plans removal:** The current MATSim implementation has a tendency to retain similar, i.e., correlated, plans when the number of plans has grown beyond J , because the current default plans remover deletes the plan with the lowest score, which is also typically most different from other plans. As a result, normally only very similar plans—with very similar scores—remain in the choice set.

These three issues can lead to biased behavior, which would have consequences for economic evaluation. Possible solutions for these shortcomings are discussed in Section 49.2, and again, from a different angle, in Section 97.3. For the rest of this chapter, it will be assumed that the above issues are solved, and that a consistent solution has been found for the system states before and after the policy change. However, the following text briefly discusses possible impacts of the above issues on policy appraisal results, to facilitate better understanding.

51.2 Valuing Human Behavior at the Individual Level

Following de Jong et al. (2007), a major advantage of the agent-based approach is a seamless integration of (i) forecasting behavioral changes as a reaction to changes in the system, and (ii) the subsequent economic evaluation. In this section, it is shown how estimated agent-specific preferences, which determine behavior, can directly be used for deriving individual VTTSs and how they need to be modified for running a MATSim simulation to obtain individual utility differences resulting from a policy change. The next Section 51.3 will then focus on how these individual utility changes can be used to derive an indicator of overall welfare change for the considered population.

51.2.1 The Utility of Time

The MATSim scoring function of plan (= alternative) i consisting of $q = 0..N - 1$ activities and trips has been introduced in Chapter 3 in the following form:

$$U = \sum_q U_{act,q}(t_{dur,q}, \dots) + \sum_q U_{trav,q}(t_{trav,q}, \dots), \quad (51.1)$$

where monetary payments (e.g., tolls) are included in $U_{trav,q}$ and the index i was dropped for notational convenience.¹

An approximate argument about optimal time allocation can be made as follows: Assume the constraint $T - \sum_q t_{dur,q} - \sum_q t_{trav,q} = 0$, i.e., that the time per day is limited by $T = 24h$, during which all trips and activities need to be completed. Let us now also assume that all travel times are fixed; i.e., we ignore the possible optimization from departure time or mode switches and concentrate on the activity time allocation problem. Optimizing under this constraint leads to the Lagrangian

$$L = \sum_q U_{act,q}(t_{dur,q}, \dots) + \sum_q U_{trav,q}(t_{trav,q}, \dots) + \mu \cdot (T - \sum_q t_{dur,q} - \sum_q t_{trav,q}), \quad (51.2)$$

where μ is the Lagrangian multiplier corresponding to the time constraint.²

Solving the optimization problem leads to

$$0 \stackrel{!}{=} \frac{\partial L}{\partial t_{dur,q}} = U'_{act,q}(t_{dur,q}, \dots) - \mu \quad (\forall q) \quad (51.3)$$

and the time constraint equation from above, where $U'_{act,q} := \partial U_{act,q} / \partial t_{dur,q}$. Equation (51.3) states that, at the optimum and without further constraints, the $t_{dur,q}$ need be selected for all activities q such that all $U'_{act,q}(t_{dur,q}, \dots)$ are the same and equal to μ .

Equation (51.2) can also be seen as a linearized version of the indirect utility function; for example, reducing travel duration by Δt_q affects not only $U_{trav,q}$, but will also lead to a utility change of $\mu \cdot \Delta t_q$ from the constraint, which can be interpreted as the linearized utility effect of spending that time otherwise.³ In consequence, the **marginal utility of time spent traveling** reads

$$\frac{\partial L}{\partial t_{trav,q}} = U'_{trav,q}(t_{trav,q}, \dots) - \mu. \quad (51.4)$$

μ is the **marginal utility of time as a resource**—the marginal utility generated by increasing T , i.e., by making the day longer than 24 hours. The marginal utility of time spent traveling is thus determined by μ , modified by “any enjoyment or dislike of the travel itself” (Small, 2012).

To get a handle on the MATSim utility function in Equation (51.1), μ and $U'_{trav,q}$ need to be obtained separately: μ in order to calibrate $U'_{act,q}$ as in Equation (51.3) and $U'_{trav,q}$ to calibrate the direct utility of time spent traveling, the offset to the marginal utility of time as a resource. This will be further discussed in Section 51.2.4.

51.2.2 The Utility of Money

Time allocation theory (DeSerpa, 1971; Jara-Díaz and Guevara, 2003) makes a similar argument for money, with a budget constraint similar to the time constraint. Just as the time constraint leads to a marginal utility of time as a resource, the budget constraint leads to a marginal utility of money as a resource.

¹ Strictly speaking, at this point, it would make more sense to stay with the scores S that MATSim generates. Section 51.2.5 discusses the relation between MATSim scores S , systematic utility V and total utility U in more detail. However, since the following text uses terms like “marginal utility of time” or “marginal utility of money”, equations are also noted using U instead of S .

² This should not be confused with the scale parameter from discrete choice theory; here, to be consistent with time allocation theory, μ represents the marginal utility of time as a resource and corresponds to β_{dur} in Chapter 3.

³ A reminder: the indirect utility function describes utility as a function of the value of the constraint that emerges when, for each value of the constraint, utility is maximized.

However, MATSim does currently not include such a monetary budget constraint. It is also questionable whether it should be introduced: the typical theoretical argument assumes the possibility of increasing one's income by working more hours. It is questionable if this functions in European countries, where work contracts typically include a fixed number of working hours, which cannot easily be changed. Hence, an alternative derivation of the marginal utility of money is necessary.

Assume that $U_{trav,q}$ includes a change in the monetary budget, $\lambda \cdot \Delta m$, e.g., invoked by fares or tolls. Then

$$\frac{\partial U}{\partial m} = \frac{\partial U_{trav,q}}{\partial m} = \lambda, \quad (51.5)$$

that is, reducing the monetary budget by Δm reduces the utility by $\lambda \cdot \Delta m$. We will therefore interpret λ as the **marginal utility of money**.⁴ Taking the first derivative of L with respect to m would lead to the same result.

In contrast to the marginal utility of time above, we do *not* break down the marginal utility of spending money for travel into a marginal utility of money as a resource, and an offset for spending money on a particular purpose (for an example of this decomposition, see, e.g. Munizaga et al., 2008). Because there is no monetary budget constraint, there is also no neutral Lagrange multiplier that would give the marginal utility of money as a resource.

This, however, leads to the problem that if there are multiple monetary channels, they may have different marginal utilities of money. For example, the marginal utility of toll payments is larger than the marginal utility of payments for fuel—i.e., people find it less irritating to pay for fuel than to pay tolls (see, e.g., Vrtic et al., 2008). That is, each monetary channel, such as fuel cost, toll, public transport fare, or a toll refund, may lead to different preference estimates.

To our knowledge, there is no best solution to this problem in the literature. For the time being, we work with forcing all alternatives' cost-related parameters to a uniform value in preference estimation. However, choice modelers typically avoid limiting the model's degrees of freedom in this way, since it suppresses some information contained in the data.⁵ It is therefore often impossible to obtain necessary parameter estimates from the literature. Where raw data is available, the same model can be re-estimated with a uniform marginal utility of money across alternatives (see, e.g., Kichhöfer et al., 2011; Tirachini et al., 2014).

Also, Small (2012) points out that the “neutral” marginal utility of money as a resource is difficult to estimate; for example, it is *not* the marginal utility of income. As an alternative research avenue, we could hypothesize that a measure's monetary channels are included in the choice experiment. For example, a travel time improvement in a value-of-time study could come together with a hypothetical income tax increase, *or* with a hypothetical toll. A rudimentary version of this actually takes place in Switzerland, where large infrastructure investments are bundled with tax increases that pay for them before they are put to public vote (see, e.g., BAV, 2013).

51.2.3 Value of Time

The **VTTs of trip q** is now defined as the marginal utility of time spent traveling (Equation (51.4)), divided by the marginal utility of money (Equation (51.5)), i.e.,

$$VTTs_q = \frac{\partial L / \partial t_{trav,q}}{\partial L / \partial m}, \quad (51.6)$$

⁴ This constant, potentially person-specific, implies that income effects (Herriges and Kling, 1999; Daly et al., 2008; Dagsvik and Karlström, 2005; Jara-Díaz and Videla, 1989) do not play a role, i.e., that changes in expenses resulting from transport policies are not strong enough to change λ . In microeconomic theory, λ is the usual variable for the marginal utility of money and corresponds to β_m in Chapter 3.

⁵ J. de Dios Ortúzar, personal communication.

where we are using the indirect utility function since we assume that the traveler compares optimal allocations before and after the change.

With $\partial L / \partial t_{trav,q} = U'_{trav,q} - \mu$ from Equation (51.2) one obtains

$$VTTS_q = -\frac{U'_{trav,q}}{\lambda} + \frac{\mu}{\lambda}, \quad (51.7)$$

μ / λ is sometimes called the value of time as a resource.

51.2.4 From Estimated to MATSim Parameters

As stated above, most value of time studies do not separately estimate μ , λ , and $U'_{trav,q} (\forall q)$. Assume that an MNL estimation of behavioral parameters from a mode choice survey between car and PT uses the following utility functions:

$$\begin{aligned} U_{car,q} &= \hat{\beta}_{trav,car} \cdot t_{car,q} + \hat{\beta}_m \cdot \Delta m_{car,q} \\ U_{pt,q} &= \hat{\beta}_0 + \hat{\beta}_{trav,pt} \cdot t_{pt,q} + \hat{\beta}_m \cdot \Delta m_{pt,q}, \end{aligned} \quad (51.8)$$

where $t_{car,q}$, $t_{pt,q}$, $\Delta m_{car,q}$ and $\Delta m_{pt,q}$ are, respectively, travel times and monetary costs in the different modes, and $\hat{\beta}_x$ are the corresponding parameter estimates. As explained in Section 51.2.2, $\hat{\beta}_m$ (the same as λ above) is assumed to be the same for all modes, or more precisely, for all types of expenditure.

According to Equation (51.4), the marginal utility of time spent traveling needs to be split into two components:

1. The marginal utility of time as resource, which needs to be used for $U'_{act}(t_{dur,q}, \dots) (\forall q)$ in Equation (51.3).
2. The direct marginal utility of time spent traveling, which needs to be used for $U'_{trav,q}(t_{trav,q}, \dots)$.

We do not know of any good way to perform this split; Kickhöfer et al. (2011) and Kickhöfer (2014) use the least negative $\hat{\beta}_{trav,mode}$ for μ (i.e., $\hat{\beta}_{dur}$) and then re-calculate all other direct marginal utilities of travel time relative to that. As indicated in Section 3.4 of this book, this is currently the preferred procedure.

51.2.5 From Simulation Output to Evaluation

At the end of the simulation run, each agent n has a number of plans $i = 1..J$, each of them associated with a score $S_{n,i}$, computed according to Equation (51.1). For economic evaluation, the question arises how to aggregate these $S_{n,i}$ into an agent-value S_n , which can then be interpreted as a utility U_n . Possibilities include using:

- the logsum of the agent's plans scores, i.e., $\ln \sum_i e^{S_i}$
- the score of the agent's last executed plan,
- the average of the agent's plans scores, or
- the highest score of the agent's plans.

51.2.5.1 Using the Logsum of the Agent's Plans Scores

In literature, the logsum term

$$\text{logsum}_n = \ln \sum_i e^{V_i}$$

has been proposed for applied welfare analysis with Discrete Choice Models (Small and Rosen, 1981; de Jong et al., 2006; Kohli and Daly, 2006; de Jong et al., 2007). Under the assumption of a correctly specified model and choice set, the logsum term represents the EMU (Expected Maximum Utility) for a user with several options $i = 1..J$ in her choice set and the systematic utility of each option i is V_i . It is the expectation value, given that a random (Gumbel-distributed) ε_i is added to each V_i , and that the individual chooses the alternative with the highest $U_i = V_i + \varepsilon_i$.⁶ In this interpretation, the (expected/average) MATSim score S_i is equated with the systematic part of the utility V_i .

However, as described in the previous Section 51.1, the use of MATSim as choice set generator yields issues with incompleteness of the choice set and with similarity of daily plans. In the current MATSim implementation, the maximum error occurs when all plans are copies of the best plan, rather than a diversity of plans. An upper bound of this error can be approximated as follows. Without loss of generality, assume that $i = 1$ is the plan with the largest systematic utility. Then

$$\logsum_n = EMU_n = \ln \sum_{i=1}^J e^{V_i} \leq \ln \sum_{i=1}^J e^{V_1} = \ln(J \cdot e^{V_1}) = \ln J + \ln e^{V_1} = V_1 + \ln J .$$

At the same time, obviously

$$\logsum_n = EMU_n = \ln \sum_{i=1}^J e^{V_i} \geq \ln e^{V_1} = V_1 .$$

Overall,

$$V_1 \leq \logsum_n \leq V_1 + \ln J .$$

That is, for a choice set with I alternatives, the true logsum value lies between the systematic utility of the best option, V_1 , and $V_1 + \ln J$.

51.2.5.2 Using the Score of the Agent's Last Executed Plan

Using, for each agent, the logsum over the scores of all plans implies that all these plans are valid behavioral choices. An alternative would be to simply use the score of the last executed plan. The behavioral interpretation consistent with this procedure is that there is no additional relevant randomness beyond what MATSim generates intrinsically. There has been no systematic work in this direction in the MATSim context, but such an approach might be justified in conjunction with the idea of explicitly generating the missing $\varepsilon_{n,i}$ for each person-alternative-pair n, i , then always selecting the best plan, as described in Section 97.4.6.

51.2.5.3 Using the Average of the Agent's Plans Scores

In principle, it is also possible to use

$$S_n = \frac{1}{J} \sum_{i=1}^J S_{n,i} \cdot P_{n,i} , \quad (51.9)$$

where $P_{n,i}$ is the probability of plan i for agent n . This can, however, only be justified when the choice probabilities, $P_{n,i}$, are interpreted like mixed strategies from game theory, i.e., that sampling from these probabilities is the true agent behavior. In principle, we cannot see why such an

⁶ At this point, we assume that V_i is absorbing the scale parameter.

interpretation should be plausible—except that it is statistically the same as Section 51.2.5.2 with the advantage of having less variance. Note, however, that the approach is intertwined with the choice model. If, e.g., $P_{n,i}$ is one for the plan with the highest score and zero for all other plans, then Section 51.2.5.2 and Equation (51.9) are identical.

51.2.5.4 Using the Highest Score of the Agent's Plans

Alternatively, one could simply use the highest score that the agent has in its plan. This would only make sense if true behavior is assumed to always select the plan with the highest score. Again, this should then also be expressed by the choice model, i.e., using the highest score only makes sense when the agent always selects the plan with the highest score, in which case the result becomes the same as Section 51.2.5.2 and 51.2.5.3.

51.2.5.5 More Complicated Variants

Section 49.1.2 discusses the idea that MATSim's typical choice model might be described by a mixture-of-logit model. In that model, ϵ_{ni} remains fixed per agent n and alternative i , but other attributes such as the network conditions vary from one iteration to the next. In Equation (49.5), η_{ni} denotes these random, but simulation-generated, deviations from the average conditions; let us add an index k for the iteration number, i.e., write η_{ni}^k . That is, it is postulated that a real person would know both ϵ_{ni} and η_{ni}^k , but the simulation only knows the latter (through the MATSim score). Equation (49.5) then just describes the resulting choice distribution from what MATSim often does, i.e., apply a logit model to scores that are not averaged.

At least for η_{ni}^k that are uncorrelated from one iteration to the next it is, however, clear that this will not result in optimal *average* agent behavior – the agent may be pushed towards some choice by a random fluctuation of the η_{ni}^k , but obtaining a much lower score from that choice in the average. Overall, the agent would be better off by first averaging the score of each alternative over many iterations, and then basing her choice on those scores. This goes back to the converged scores of Section 49.1.1.

Calculating benefits from a mixture-of-logit interpretation becomes thus rather involved: we postulate that the agent sees the full MATSim score, plus some private ϵ s; that she optimizes based on the sum of these two; that the MATSim simulation, however, does not know the ϵ s and thus has to sample from the logit model; but that the economic utility has to include the effect of the ϵ s although we do not know them, as in Section 51.2.5.1. Overall, thus, assigning utility values to such behavior as described by Equation (49.5) requires a better understanding of underlying behavioral rationality. Section 97.4.6 discusses this further.

51.2.5.6 Summary

Overall, there seem to be two consistent strategies to aggregate various plan scores of an agent n into one value:

- If the choice model is a logit model, then using the logsum term over all plan scores as the agent's utility U_i is consistent with the choice model.
- If the choice model is such that the plan with the highest score is selected, then using that score as the agent's utility U_i is consistent with the choice model.

In both cases, the choice model needs to be consistent with the behavioral assumption about the agent, i.e., in the first case it needs to be assumed that the model does not know the true agent choice beyond the choice probabilities and the model system thus has to repeatedly sample from these probabilities. In the second case it needs to be assumed that the randomness has already been “frozen” into the score computation (see Section 97.4.6) and the agent thus selects the plan with the highest score.

In both cases, the calculated individual score differences that result from a policy measure can be directly used in order to identify winners and losers.

Some economists claim that the modeler's task of providing information for decision support ends at this point (Ahlheim and Rose, 1989). However, in practice, some (monetary) valuation of the resulting behavioral changes is often required. The next section reviews different possibilities to monetize and aggregate individual utility differences in the MATSim context.

51.3 Aggregating Individual Values

After having obtained the individual changes in terms of utility, it is often necessary to convert these utility changes into monetary terms for economic evaluation, e.g., in BCA. Unfortunately, no "correct" monetization or aggregation approach exists for individual utility differences. This is reflected by the ongoing discussion⁷ between transport policy appraisal experts:

1. The first stream argues in favor of a consistency in values used in demand modeling and appraisal (Grant-Muller et al. (2001, p.255), Bickel et al. (2006, p.S4 and p.S8), and Proost⁸). Values from literature should only be used if behavioral model values are not available. These researchers are, however, aware that this procedure potentially limits the comparability of projects in different regions of the same state, or in different member states of the EU (European Union). In consequence, additional indicators such as absolute time savings per income group should also be reported to address equity issues.
2. In contrast to the above, Mackie and Worsley (2013, p.12) state, that in the United Kingdom, "standard [VTTS] values per minute would be used across incomes, modes and regions. Therefore, their practice is to use behavioral information for modeling but standard values for appraisal." Also Daly (2013) distinguishes between "valuation", i.e., people's willingness-to-pay (or accept) for marginal changes, and "appraisal", i.e., what these changes are worth from a societal point of view.
3. Fowkes (2010), OECD (2006), and Gühnemann⁹ argue slightly differently, but in the same direction: modeling and evaluation should be based on the best heterogeneous preferences available; in the evaluation, additional weights should be introduced, e.g., to counter the effect of decreasing marginal utilities of money, or increasing VTTS with income, respectively. These weights would, thus, define the underlying equity concept of the appraisal method.
4. However, as Ahlheim and Rose (1989) point out, no approach to empirically determine these weights is available without assuming some arbitrary a-priori specification. In consequence, every interpersonal comparison of utility changes requires some normative decision and the weights need therefore to be determined on a political level.

One goal of this section 51.3 is to show the impact of a possible integration between behavioral modeling and economic evaluation in the same agent-based framework. First, a conversion into *income equivalents*, and second, a conversion into *time equivalents* (possibly followed by some conversion into money terms).¹⁰ The choice of the procedure depends on a (normative) decision whether one *EUR* or one *h* should be valued equally across individuals. It is, therefore, important

⁷ A similar overview on this discussion is given by Börjesson and Eliasson (2014).

⁸ S. Proost, personal communication.

⁹ A. Gühnemann, personal communication.

¹⁰ Kickhöfer (2014) shows that the choice of the monetization and aggregation procedure can have major impact on the results when heterogeneity is assumed in user preferences.

that decision makers and modelers who deal with economic evaluation understand the possible effects of that choice; simply going with the most common approach may not be advisable.

51.3.1 Income Equivalents

Basic Approach The most common approach used in welfare economics to convert utility changes into money terms is to calculate the monetary amount ΔY_n that one would need to give or take from individual n to offset the impact of the policy on the utility level ΔU_n . According to Equation (51.1), it is calculated as

$$\Delta Y_n = -\frac{\Delta U_n}{\lambda_n} . \quad (51.10)$$

Note that the marginal utility of money, λ_n , might be person-specific, e.g., dependent on the person's income.

The monetary amount $-\Delta Y_n$ from above represents individual Consumer Surplus. Its absolute value is, in the absence of income effects (see Footnote 4 in Section 51.2.1), equal to the Compensating Variation and the Equivalent Variation (Daly et al., 2008). The overall welfare change ΔW for the population with individuals $n = 1..N$ is then calculated by

$$\Delta W = -\sum_{n=1}^N \Delta Y_n . \quad (51.11)$$

Equity The above approach is often criticized for equity reasons: if the marginal utility of money is—in the behavioral model—assumed to decrease with income, and these values are directly (without additional weights) used in economic evaluation, rich people will have a stronger impact in the evaluation process than poor people. In turn, this might lead, e.g., to investments in expensive high-speed trains on major corridors rather than affordable train services for everyone. In terms of equity and public acceptance, such specification in the appraisal method might not be desirable. To counter this effect in economic evaluation, the use of standard or equity values is proposed in the literature. In this context, Jara-Díaz (2007, p.106ff) introduces the *social utility of money* and the *social price of time*. For a more general overview of possible solutions how to address equity issues, see Rizzi and Steinmetz (2015).

A rather ad-hoc but simple possibility is to replace the person-specific marginal utility of money, λ_n , with a population average,

$$\bar{\lambda} := \frac{1}{N} \sum_n \lambda_n , \quad (51.12)$$

and then

$$\Delta Y_n = -\frac{\Delta U_n}{\bar{\lambda}} . \quad (51.13)$$

Following the argument by Fowkes (2010), OECD (2006) and Gühnemann et al. (2011) mentioned above (Item 3), this would be one particular way to introduce the necessary weights. Alternatively, one could think of fixing the social weight of every person to 1.0, and derive the social price of all attributes included in the generalized costs from there (Jara-Díaz, 2007, p.108f).

51.3.2 Time Equivalents

Another option to derive a monetary measure of welfare changes is composed of two steps: First, a conversion of individual utility changes into *equivalent hours of time as a resource* (Jara-Díaz et al., 2008; Mackie et al., 2001). This would be the number of hours ΔT_n that one would need to give or

take from individual n to offset the policy impact on the utility level ΔU_n . Second, a monetization of the resulting numbers through an arbitrary conversion factor, i.e., the monetary value of one hour for the individual or for society.

In the MATSim sense, one could first calculate the corresponding time equivalent by

$$\Delta T_n = -\frac{\Delta U_n}{\mu_n}. \quad (51.14)$$

Similar to the marginal utility of money, also the marginal utility of time as a resource, μ_n , might be person-specific.

One option would be simply provide time equivalents, i.e., the BCA would return time equivalents per invested monetary unit. In many situations, however, it is desirable to convert all impacts of a policy into monetary terms, i.e., to compute,

$$\Delta Y_n = \alpha_n \cdot \Delta T_n, \quad (51.15)$$

and to compare ΔY_n with investment or changes in external costs. The following options are then possibilities for α_n :

- The obtained time equivalents ΔT_n could be converted in monetary terms using the person-specific resource values of time, i.e.,

$$\alpha_n = \frac{\mu_n}{\lambda_n}. \quad (51.16)$$

This would obviously result in the same monetary amount as the income equivalent approach from Equation (51.10).

- Following Mackie and Worsley (2013), one could argue that the resource value of time should be the same for every individual, and, thus, use some average value for monetization, e.g.,

$$\alpha_n \equiv \bar{\alpha} = \frac{1}{N} \sum_{n=1}^N \frac{\mu_n}{\lambda_n}.$$

- As another alternative, one could average over the marginal utility of money only, i.e.,

$$\bar{\lambda} = \frac{1}{N} \sum_n \lambda_n$$

and then

$$\alpha_n = \frac{\mu_n}{\bar{\lambda}}. \quad (51.17)$$

This would highlight that some persons are more pressed for time than others, while, at the same time, using an equal value for the marginal utility of money. Clearly, this gives the same result as Equations (51.12) and (51.13). It does, however, lend itself to a clearer interpretation: first, all utility differences are converted to a comparable scale, i.e., time as a resource (Equation 51.14). Then, these times are converted to a monetary scale, using a conversion factor which includes the pressure for time (i.e., the person-specific μ_n) but assumes an average marginal utility of money.

In all cases, the overall welfare change ΔW for the population with individuals $n = 1..N$ is then calculated identically to Equation (51.11), i.e., by $\Delta W = -\sum_{n=1}^N \Delta Y_n$.

51.3.3 *Income vs Time Equivalents: Discussion*

The sections above show how to monetize and aggregate individual utility differences through income equivalents or time equivalents. To summarize:

- Income equivalents put emphasis on the individual willingness-to-pay, whereas time equivalents focus on time pressure.
- The aggregation of income equivalents yields the overall equivalent monetary cash flow that would be generated by the project for the population considered. That is, one *EUR* is valued equally across individuals.
- The aggregation of time equivalents yields the overall equivalent lifetime hours that would be generated by the project for the population considered. That is, one hour of lifetime is valued equally across individuals.
- A monetization of time equivalents using person- and activity-specific resource values of time leads to the same total benefit as directly aggregating income equivalents.
- A monetization of time equivalents using some average value of time as a resource, therefore generally leads to a different total benefit than directly aggregating income equivalents. Such an approach maintains the equal value for one *h* of lifetime.

51.3.4 *Conclusion and Recommendations*

Scoring Function A correct scoring function is central to correct MATSim functioning. The mathematics and understanding of that scoring function need to be derived from time allocation theory in economics. In particular, any marginal utility of travel time needs to be split into the marginal utility of time as a resource (μ in the text above, and β_{dur} in Section 3.4) and an additional direct marginal utility of time spent traveling ($U'_{trav,q}$ in the text above, and $\beta_{trav,mode,q}$ in Section 3.4).

Since most discrete choice models estimate the sum of these two, definition is required about how to split up this sum. A somewhat ad-hoc way to achieve this is to find the mode with the largest (= least negative) marginal utility of time and use that value for the marginal utility of time as a resource. That reference mode's direct marginal utility of time spent traveling is then zero; all other modes' direct marginal utilities of time spent traveling are relative to that of the reference mode.

If one is interested in monetization, i.e., converting utility values into monetary terms, then additionally the marginal utility of money as a resource (λ in the text above, and β_m in Section 3.4) needs to be known. Our current approach to obtain an approximation to λ is to force all monetary preferences in the estimation of a choice model to a unique value. If this is not possible, then one has to make a normative decision which monetary channel is considered most “neutral”, i.e., most similar to an “unearned income” channel.

Choice Model and Score Aggregation MATSim agents normally have more than one plan; each plan has a score. There are two consistent approaches to come up with a utility value from those scores:

- Using a MNL choice model that makes probabilistic draws from those plans using their scores: The correct aggregation is then the logsum of all scores.
- Using a choice model that selects the plan with the highest score: The correct aggregation is then to use the score of that plan.

In both cases, the result is the total utility U of the choice set. In the first case, the logsum term includes an expectation value of the randomness, typically denoted by ε . In the second case, all randomness, if any, needs to be “frozen” into the alternatives, and included into the computation of the score.

Monetization Individual utility differences resulting from a change in the transport system can be converted into monetary terms by dividing them by λ_n . The result is the change in individual user benefit. Aggregating these individual benefits provides an indicator for the overall welfare change. Since λ_n may vary among agents, e.g., according to their incomes, such approach will put a higher weight on people with small λ_n , typically those with large incomes. An alternative is to use an average $\bar{\lambda}$ for this conversion, even when the behavioral model (= the scoring function) uses person-specific λ_n .

Acknowledgements

The authors are grateful to G. Liedtke (DLR Berlin) who provided very helpful and detailed feedback after reading two rather different versions of this chapter. The authors would also like to thank C. Winkler (DLR Berlin) for his useful comments, in particular on the use of the indirect utility functions for economic evaluation. Finally, the authors are very thankful for the discussions with F. B. Birke (DIW Berlin) who formalized the possible decomposition of the marginal utility of money. The responsibility of any remaining errors stays with the authors.

PART IV

Scenarios



CHAPTER 52

Scenarios Overview

Marcel Rieser, Andreas Horni and Kai Nagel

This last book part summarizes MATSim scenarios, as located on the map in Figure 52.1 and listed at <http://matsim.org/scenarios>.



Figure 52.1: Locations with known MATSim scenarios. Most of them are described in this book.

How to cite this book chapter:

Rieser, M, Horni, A and Nagel, K. 2016. Scenarios Overview. In: Horni, A, Nagel, K and Axhausen, K W. (eds.) *The Multi-Agent Transport Simulation MATSim*, Pp. 367–368. London: Ubiquity Press. DOI: <http://dx.doi.org/10.5334/baw.52>. License: CC-BY 4.0

Although there are real-world scenarios based on free and public data such as the Santiago or Cottbus scenarios (Chapters 84 or 66), many scenarios are not public, due to data privacy issues. However, knowing about general methods and approaches adapted for scenario creation and understanding problems faced during these processes might significantly support and encourage the building of new scenarios. Each of the following chapters provides information on study area, population and demand generation, activity locations, network, simulated modes, calibration and validation, achieved results, and associated projects. Further topics involve where to find more information and where/when emphasis is put on certain scenario specialties—be it parsimonious data usage procedures, special modules used, or special modes simulated (such as the parataxis in the Gauteng scenario). Some scenarios have been used for years, with ongoing further development. We target the latest version when reporting.

Different levels of MATSim involvement are possible. For some regions and projects, MATSim is, for example, used only for traffic assignment, where for others, the complete demand is endogenously handled. Couplings with other forecasting models for transport demand generation have been successfully applied, like the coupling with TASHA (Travel Activity Scheduler for Household Agents) for Toronto, or the combination of MATSim with the Tel Aviv activity-based transport model.

CHAPTER 53

Berlin I: BVG Scenario

Andreas Neumann

The BVG is Berlin's main public transport company, running virtually all services, with the exception of the S-Bahn urban rail system. This includes bus services, the subway network, the largest tram network in Germany and ferry services. The bus network consists of 149 different lines, 6468 directed stops and a vehicle fleet of 1316 buses (BVG, 2012). In total, about 937 million trips were served by BVG in 2012, 41% of them by bus.

With the opening of the new Berlin and Brandenburg BER international airport, Berlin expects major travel demand changes; importantly, the existing airport Tegel, now exclusively served by BVG-operated buses, will close. BVG thus had substantial interest in a new Berlin area transport model. To deal with these changes, the model not only had to provide a base for future regional transport system planning, but also had to supply detailed information about different user groups' passenger flows. Such user group-specific analyses were very important for BVG in providing a platform for their future business strategies; thus, an agent-based model was specifically requested. Two scenarios were required, one for the year 2008 (actual state), and one for the year 2015 (prediction). To meet the above needs, PTV (2013), Senozon (2013) and VSP (2012) at TU Berlin offered a combined model consisting of both a static macroscopic model built with VISUM, as well as an integrated, activity-based demand and dynamic traffic assignment model, built with MATSim. During the project, efforts were made to base both models on the same data sources and to ensure that both modeling processes interacted with each other to allow data exchange.

The model contains about 115 000 links, about 15 000 directed stops, about 6 million agents, and 539 public transport lines operated by BVG and other Berlin and Brandenburg state companies. Besides motorized individual traffic and public transport, the model also considers biking and walking. For a more in-depth description of the model, its generation and its calibration, the reader is referred to the work of Neumann et al. (2014). The model has extensively been used by Neumann (2014, Ch. 7/8) for the development of the minibus module presented in Chapter 17.

How to cite this book chapter:

Neumann, A. 2016. Berlin I: BVG Scenario. In: Horni, A, Nagel, K and Axhausen, K W. (eds.) *The Multi-Agent Transport Simulation MATSim*, Pp. 369–370. London: Ubiquity Press. DOI: <http://dx.doi.org/10.5334/baw.53>. License: CC-BY 4.0



Figure 53.1: The city of Berlin and its transit network.

Berlin II: CEMDAP-MATSim-Cadyts Scenario

Dominik Ziemke

To correctly model initial demand properties not included in MATSim iterations in specific studies (i.e., activity choice), suitable data are needed. Travel diaries containing departure times, mode choice decisions and activity locations are widely used. However, much of this data source content, particularly location information, is considered sensitive in terms of data privacy legislation and thus increasingly difficult to obtain and process in many areas (e.g., in Germany and the United States; Ziemke et al., 2015).

The *Berlin II scenario* (also referred to as the *CEMDAP-MATSim-Cadyts scenario* according to the applied models in its setup), is the outcome of an alternative approach relying exclusively on freely available or easy-to-obtain input data. All of these data do not rely on individual trajectories, but instead on “anonymous” data that is aggregated so much that the data providers are no longer concerned about privacy issues.

The starting point for this scenario is a publicly available commuting matrix containing homes and workplaces of workers with social security on the municipality level. Based on this information, it is possible to model morning and evening commuting peaks.

To obtain a full-population demand representation, two further major modeling steps are required. First, in cases like the Berlin case, see below, where the commuter matrix spatial resolution is quite coarse, higher resolution O-D information is necessary. Second, a procedure is needed to model secondary activities, i.e., all other activities beyond home and work.

The importance of the first step becomes obvious when looking at the German case; here, the whole city of Berlin, with 3.4 million inhabitants, is represented by exactly one zone (Bundesagentur für Arbeit, 2010). In the United States, commuting matrices are typically available only on a county-to-county level. Since such location-aggregation-based matrices may become the rule, rather than the exception, in privacy-sensitive societies, a (generalizable) method to attain O-D information at a higher resolution is needed (Ziemke et al., 2015). The standard solution would be to estimate an activity location choice model. This, however, is difficult if no trip data to estimate

How to cite this book chapter:

Ziemke, D. 2016. Berlin II: CEMDAP-MATSim-Cadyts Scenario. In: Horni, A, Nagel, K and Axhausen, K W. (eds.) *The Multi-Agent Transport Simulation MATSim*, Pp. 371–372. London: Ubiquity Press. DOI: <http://dx.doi.org/10.5334/baw.54>. License: CC-BY 4.0

the model is available. O-D matrix estimation studies (van Zuylen and Willumsen, 1980) suggest that traffic counts may be used to make an initially rough O-D matrix more appropriate for a region. As MATSim is not based on O-D flows, but on full daily plans, the issue comes down to whether a procedure exists to update these initial full daily plans using traffic counts. In the approach used to create the Berlin II scenario, a procedure proposed by Flötteröd et al. (2011) and implemented in the software Cadyts—explained in Chapter 32—is applied for this task. Specifically, random draws of possible home and work locations within the home or work municipality given by the commuter matrix are made. Various MATSim plans, each containing one pair of home and work locations, are created for each agent. Then, the Cadyts calibration procedure is applied within the iterative MATSim simulation to select plans and locations more likely to occur with given traffic counts.

As stated above, however, full daily plans (as opposed to mere home-work-home commuting patterns) are needed. Therefore, the second modeling step, the modeling of secondary activities for each individual in the region, needs to be addressed. For the Berlin II scenario, CEMDAP (Comprehensive Econometric Microsimulator for Daily Activity-Travel Patterns (Bhat et al., 2008)) is used to generate initial complete daily plans for each individual. On one hand, however, no CEMDAP parameter set is available for Berlin. On the other hand, and more importantly, one major goal of the study creating the Berlin II scenario was to show its generalizability (Ziemke et al., 2015). So, the model parameters of CEMDAP estimated for the Los Angeles region (the estimation context) are retained and then used to generate initial plans for individuals in Berlin (the application context in the current paper), based on Berlin demographic data.

To sum up, home and work municipalities are taken from the commuter matrix. Within these municipalities, a set of (more precisely spatially defined) potential home and work locations are randomly chosen for each agent. Full daily plans incorporating the various potential locations of each agent are generated with CEMDAP, based on a parameter set from another region.

Then, the Cadyts calibration procedure is used to select those initial full daily plans most consistent with Berlin traffic count data. In other studies, Cadyts has already been applied to update route choice predictions, both for car (Flötteröd et al., 2011a) and for public transit (Moyo Oliveros and Nagel, in press). However, it has not been used to update full daily activity-travel plans, as it was in the procedure that created the Berlin II scenario.

The Berlin II scenario is thus an activity-plan-based MATSim transport model for Berlin based exclusively on freely, or readily, available data. If a commuter matrix, some basic population demographics, and traffic counts (or, theoretically, another suitable data source on which to run the calibration procedure) are available for a particular regional context, the approach used to create the Berlin II scenario can be transferred to that other context. In fact, the Berlin II scenario itself should be seen as a *transferred model*, because initial plans generated by CEMDAP are based on parameter estimates from another geographic region (the Los Angeles area).

Through a validation based on the Berlin 2008 SrV (System repräsentativer Verkehrsbefragungen (Ahrens et al., 2009)), an extensive, regularly-conducted travel survey, the created transport demand representation quality has been successfully tested. So far, the Berlin II scenario exists for a 1% and a 10% population sample of all persons, i.e., including workers without social security, as well as non-working people, aged 18 and above, for the study region. Currently, only motorized traffic is considered. Stability tests, showing that plausible agents' daily plans continue to be chosen when Cadyts calibration functionality is switched off, have been successfully carried out. This is a clear indication that the scenario is applicable and meaningful for policy studies.

Further improvements, like the addition of public transport and a more realistic representation of the population, are planned. Moreover, similar approaches to integrating activity-travel pattern generators (e.g., the FEATHERS model) with MATSim in transport simulation are planned.

CHAPTER 55

Switzerland

Andreas Horni and Michael Balmer

The Switzerland scenario was initially created for the project Westumfahrung (Balmer et al., 2009a) and serves as the base for the very frequently used Zürich scenario (Chapter 56).

Two main branches can be distinguished. The first, older one is based on a one-to-one translation of the Swiss population census (Swiss Federal Statistical Office (BFS), 2000); the second applies approaches from the IPF (Iterative Proportional Fitting) family, reported by Müller and Axhausen (2013, 2012); Müller (2011b,a, 2012) to generate the synthetic population.

The scenario's study area covered all of Switzerland. Due to administrative borders, no demand and supply data were available for adjoining countries, which leads to boundary effects; studies focusing on Swiss border areas are difficult.

The population was derived from the Swiss Census of Population 2000 (Swiss Federal Statistical Office (BFS), 2000). The complete Swiss population was modeled, resulting in around 7.5 million agents.

This population's home locations were given at hectare level and work locations were specified at municipality level from commuter matrices, a component of the Swiss Census of Population 2000 (Balmer et al., 2009a, p.35). A very good overview, in German, of the population generation, its initial individual demand and activity locations can be found in Meister et al. (2009). Further information is given by Ciari et al. (2008); Meister et al. (2010); Balmer et al. (2009a, 2010, 2009b).

Travel demand was basically taken from the 2000 and 2005 National Travel Surveys (Swiss Federal Statistical Office (BFS), 2006) (Swiss microcensus), although this sample substantially underestimated freight traffic and ignored cross-border traffic of non-Swiss residents. Freight traffic for Switzerland was missing at that time (except Zürich, see next chapter). Cross-border traffic was derived from mode-specific, hourly origin-destination matrices given by Vrtic et al. (2007). These were disaggregated to around 600 000 individual MATSim plans for the whole country, which contain the cross-border traffic originating *outside* Switzerland. Non-Swiss, cross-border traffic starting in Switzerland was supposed to be negligible.

How to cite this book chapter:

Horni, A and Balmer, M. 2016. Switzerland. In: Horni, A, Nagel, K and Axhausen, K W. (eds.) *The Multi-Agent Transport Simulation MATSim*, Pp. 373–374. London: Ubiquity Press. DOI: <http://dx.doi.org/10.5334/baw.55>. License: CC-BY 4.0

The activity location data set, comprising home, work, education, shopping and leisure locations, was also derived from the 2000 Swiss Census of Population and the 2001 Federal Enterprise Census (Swiss Federal Statistical Office (BFS), 2001), providing hectare level information. Facility generation was described by Balmer et al. (2009a, p.33).

For car traffic, navigation networks from Teleatlas (Tele Atlas MultiNet, 2010) and NAVTEQ (NAVTEQ, 2011) were available. The most-used network was the planning network derived from the Swiss National Transport Model (Vrtic et al., 2003).

The public transport simulation network was derived from the National Transport Model of the UVEK (Eidgenössisches Departement für Umwelt, Verkehr, Energie und Kommunikation), described by Vrtic and Fröhlich (2010).

The scenario simulated car and public transport; schedules for public transport were given at the municipality level. Fine-granular schedules were not available then, but were in preparation. The modes walk and bike were usually “teleported”.

Calibration was mainly performed for modal split and distance distributions; utility function values were set accordingly.

For validation, count data on city level, cantonal level and national level (ASTRA, 2006) were available from various sources, resulting in 600 links measured for Switzerland. An average working day (Monday to Thursday, excluding public holidays) was used for comparisons in current projects.