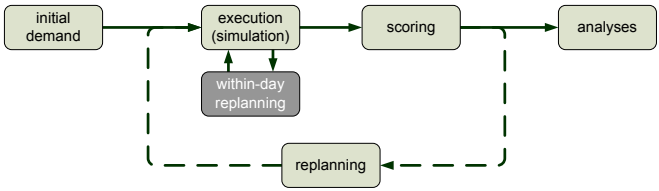
Chapter 1

Within-Day Replanning *[Dobler, Maciejewski, Nagel]*

Authors: Christoph Dobler, Michal Maciejewski, Kai Nagel



1.1 Introduction

In recent years, in transport planning and traffic management the interest in scenarios where events occur which cannot or only partially be foreseen has increased. Typical examples for events which can only partially be foreseen are taxis and car sharing. An

agent that has a taxi trip in its plan can for example not know in advance which taxi will be available when the agent needs one. When using car sharing, an agent might walk to the car sharing station and check whether a car is available or not. In case it is not, the agent could either decide to wait or adapt its plan and switch to another mode. Examples for events which cannot be foreseen at all are road accidents, terrorist attacks or disaster such as earthquakes.

Traditional simulation approaches (as used in MATSim) optimize traffic demand 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 behaviour, 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 behavioural 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.

1.2 Simulation Approaches

1.2.1 Iterative Simulation Approaches

The starting point of an iterative simulation approach, as it is used in agent-based traffic flow micro-simulations like DYNEMO (**??**), MATSim (**??**) or TRANSIMS (**??**), is the generation of an initial plan for each agent (e.g. based on census and / or travel diary data). A plan contains an agent's intended schedule of activities and the trips

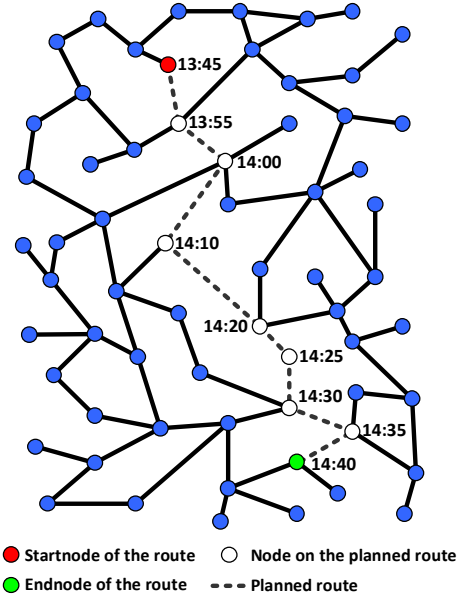
that connect them. For each activity, its type (e.g. *work*, *leisure* or *shopping*), its location and the expected start and end time are given. A transport mode and a route specify a trip. The iterative optimization process consists of three steps. First, a mobility simulation executes the plans which are then evaluated using a fitness function. Finally, agents must select plans to be executed in the next iteration. Each agent can keep several plans in its memory. Bad plans can be deleted and new plans created by cloning and adapting existing ones using information (e.g. travel times) from one or more previous iterations. The allowed adaption operations define the optimization search space (e.g. routes, location and start / end times of activities). This iterative optimization can be seen as a period-to-period replanning strategy. Since one day is a commonly used duration, it is often called day-to-day replanning strategy.

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 the best choice. First, 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.

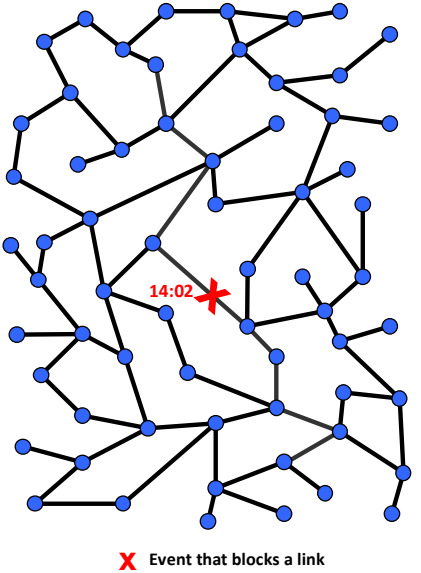
Figure **??** shows a simple example scenario where an iterative approach would produce illogical and faulty results. In Figure **??** an agent's planned route in a sample network is shown, including the times when the driver passes each node of the route. Clearly, those times are only valid if no exceptional event occurs. Figure **??** 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 **??**). When this

Figure 1.1: Exceptional event in a network

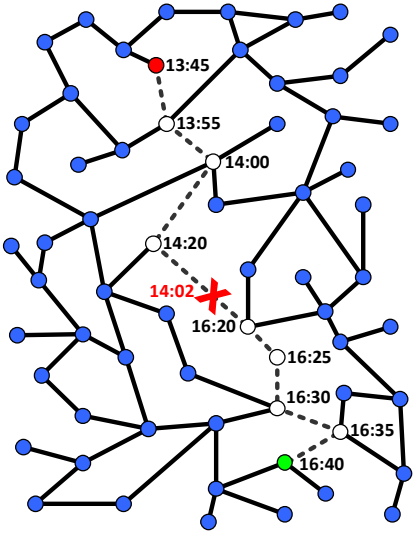
(a) Network with planned route



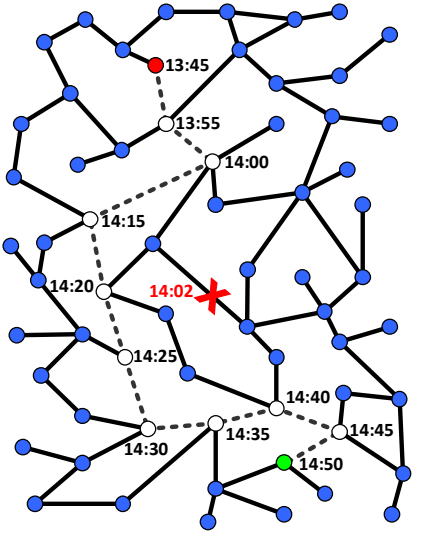
(b) Network with exceptional event



(c) Network with exceptional event and planned route



(d) Network with exceptional event and adapted route



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 **??**). 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.

1.2.2 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.

1.2.3 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.

1.3 Implementation

TODO: describe why there are two different implementations!

1.3.1 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 mobility simulation (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 taken. The most important decision for a driver is to chose 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 alight at the current stop?
- What is the ID of the vehicle to be used for a trip?

From the perspective of the agent such an approach might be called *event driven*, since the agent performs mental activity only 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 **??**.

Challenges with the above approach include:

- 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 compute plausible routes.

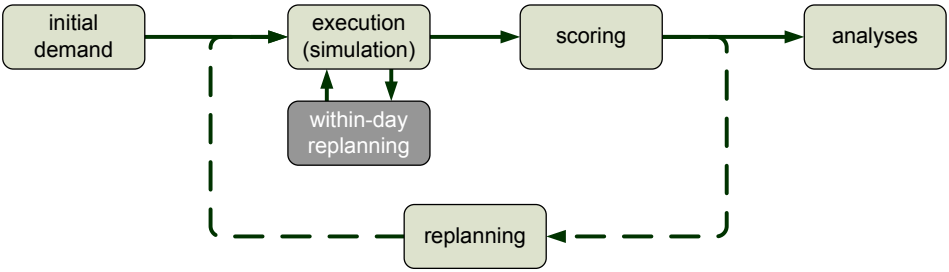
- There are situations where the capability to decide the turn at each intersection while en-route is, in fact, not needed.

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 the recommended evacuation directions change because of a change in wind direction. In other applications, evacuating agents may become stuck in unexpected congestion which may trigger en-route re-routing. This may, however, to restricted to relatively small regions, and it may be sufficient to go through such a replanning loop every, say, 300 simulated seconds.

For such applications, the plan-based approach (Section **??**) 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, and then goes through those agents and changes the future portion of their plans. This is not only conceptually easier than having every agent to answer for him-/herself, but it is also computationally more efficient since it is only called when it is triggered, and for 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, say, 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, for example for an entirely reactive taxi driver, then this points to the approach of replacing the agent.

Figure 1.2: (Iterative) within-day replanning MATSim loop [**kn**: I think this figure needs to come earlier.]



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 makes it possible without additional computational overhead that agents make decisions at the latest possible time. 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, say, random elements in terms of how far an agent plans ahead.

1.3.2 Implementation alternative 1: Plan-based Implementation

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

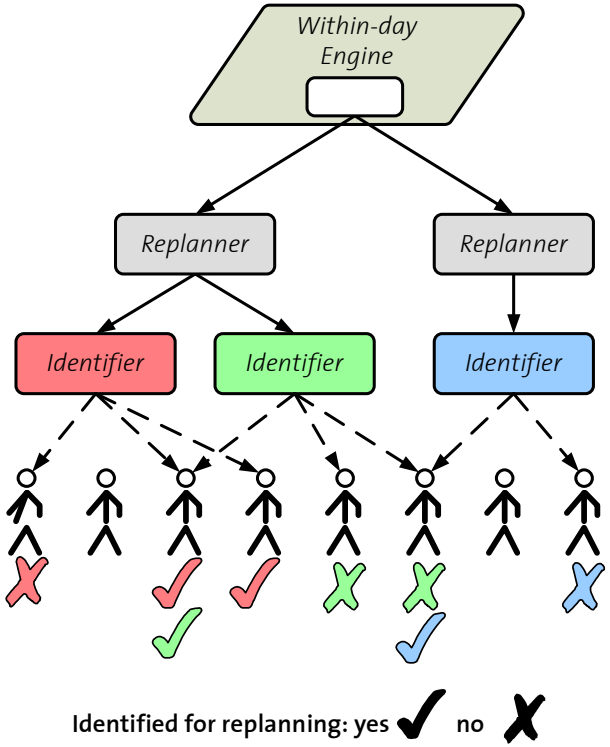
TODO: redraw this figure using the new (blue) matsim loop?

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 simulated 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 **??** 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 *Identifiers* which communicate with agents and select those agents who are given the opportunity to adapt their plans. An *Identifier* can be seen as an information-distributing unit, like a radio station or a policeman. Therefore, not every *Identifier* communicates with all agents. For example, agents at home will probably listen to the radio, but agents walking in the park will not. Each *Identifier* returns a list of agents to its superior *Replanner*, which then adapts those agents' plans.

There is a division of responsibilties between *Replanners* and *Identifiers*. 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 *Identifiers* that only agents who are currently performing a leg are replanned. In turn, *Identifiers* 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 numbers are a special case. They have to ensure that their *randon number generator* is re-initialized for every replanned agent using a deterministic value (e.g. a combination

Figure 1.3: WithinDayEngine



of the agent's Id and the current time step).

Running the *Identifier(s)* to select those agents who have to adapt their plans is performed sequentially. On one hand, an *Identifier's* runtime is typically very short and therefore no significant performance losses are expected. On the other hand, it is a robust design which cannot produce race conditions which could occur if multiple instances of an *Identifier* run concurrently. An example would be an *Identifier* 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 *Identifier*, 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 twice in the list of agents to be replanned—once added by each *Identifier* instance.

Replanner implementations are available for every possible basic change of an agent's scheduled daily plan. Any 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)
- 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 have to be adapted.

Additionally, four basic *Identifiers* 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.
- performing a trip and are going to move to another link.

Often, identifiers have to handle only a subset of the population, e.g. only male agents or agents which are currently performing a car trip. In order to prevent that the same functionality has to be implemented multiple times, so-called *AgentFilters* are introduced. Their task is to remove agents which do not meet the filter criteria from a set

of agents. Using *AgentFilters* not only avoids duplicated code but can also reduce the computation effort. An example would be two *Identifiers* which should identify only agents currently travelling in a certain part of the network. Without *AgentFilters*, each of them would have to track all travelling agents and their current positions. When this functionality is moved to an *AgentFilter*, the two *Identifiers* can share a single instance of that filter.

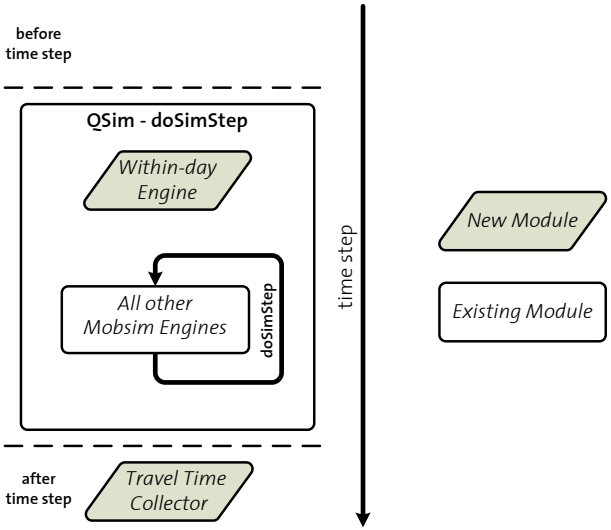
Basically, simple and re-usable functionality should be implemented as *AgentFilters* while more complex and/or decision making functionality should be part of an *Identifier*. Again, this can be depicted with an example, e.g. a scenario in which the search for a parking space is modelled. A filter can be utilized to take only agents into account which are currently travelling by car. The *Identifier* 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 set of agents.
- currently using a transport mode included in a given set.
- currently located on a link included in a predefined set.

Besides the logic that identifies agents and adapts their plans, another important part of the within-day replanning framework is code that continuously collects information and provides it to the *Identifiers* which decide based on that data whether agents are replanned or not. In a time step based approach—as it is realized by the *QSim*—collecting, analyzing and aggregating data as well as providing it can be easily realized. Figure **??** shows the structure of a *QSim's* time step. Each time step is separated into three phases—*before time step*, *do sim step* and *after time step*. During the *do sim step* phase, all registered *MobsimEngines* simulate the current time step. The *before*

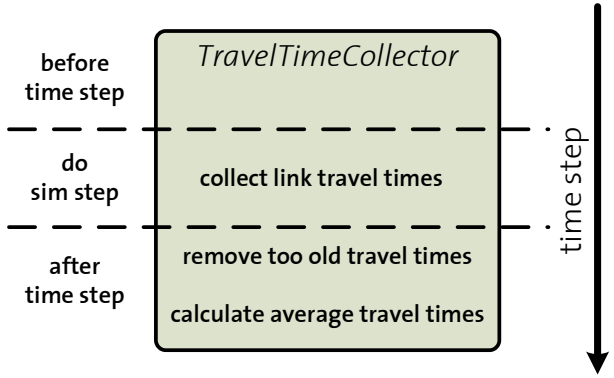
Figure 1.4: QSim time step



time step and *after time step* phases allow to execute code before respectively after the simulation of the current time step. A class can collect data like link travel times during the *do sim step* phase. Afterwards, the collected data can be analyzed and aggregated in the *after time step* phase. In the next time step, the *WithinDayEngine's Identifiers* can use that data for their decisions. The *WithinDayEngine* is always the first *MobsimEngine* which executes its *doSimStep* method. This ensures that no agent has changed its state since the *after time step* phase of the previous time step. As a result, the *Identifiers* make their decisions on actual data.

An example for such a class is the so-called *TravelTimeCollector*. Its task is to provide 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 span. A typical time span is 15 minutes; older link travel times are ignored. Duration of the specific time span has an important impact on travel times reported to the *Replanners*. On one hand,

Figure 1.5: TravelTimeCollector



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 implement features like traffic flow predictions or dynamic weighting of recent travel times based on historical data. Due to the abandonment of such features, it is very robust, even in scenarios where traffic flow conditions change dramatically.

An important aspect that has to be considered when within-day replanning is used is whether the persons' or only the agents' plans are replanned. In this context, a person is a global object which exists during an entire simulation run. An agent represents a person in the micro-simulation part of an iteration. Moreover, a person has several plans in its memory, an agent only a single one. By default, an agent's plan is only a copy of the plan of the person which is represented. Only the score of the person's plan is updated by MATSim's scoring module.

In a first one, within-day replanning is used to simulate agents' parking search. A per-

son's plan contains the location where a free parking space is expected. However, if the agent realizes in the micro-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 but maybe not in another one where the initially selected parking spot is available.

Creating agents' initial routes using within-day replanning would be an example for an application where persons' plans have to be adapted. By default, creation of those routes is a step performed on person level before the micro-simulation is started for the first time. When this task is moved into the micro-simulation and is performed during their runtime, the created routes have to be still stored in the person's plan. Otherwise they would not be available for later iterations.

The capabilities of this within-day replanning implementation are shown and discussed by **?** based on two sets of conducted experiments. The first set is based on a model of Zurich city. There it is assumed that the capacities of several arterial roads in the city center are drastically reduced during the morning peak. Travelling agents are given the opportunity to bypass the resulting traffic jams by adapting their routes using within-day replanning. The second set of experiments uses within-day replanning to create agents' initial routes. The results are compared to runs where those routes are created before the simulation is started and without having traffic flow information available.

1.3.3 Implementation alternative 2: Replacing the agent

Chapter 2

Discrete Choice Modeling Perspective on MATSim *[Flötteröd]*

Authors: Gunnar Flötteröd