DISS. ETH NO. 18034

EFFICIENT ALGORITHMS FOR THE MICROSIMULATION OF TRAVEL BEHAVIOR IN VERY LARGE SCENARIOS

A dissertation submitted to

ETH ZURICH

for the degree of

Doctor of Sciences

presented by

DAVID CHARYPAR

Master of Science in Computer Science, ETH Zurich

born 03.02.1978

citizen of
Rudolfstetten, AG

accepted on the recommendation of

Prof. Kay W. Axhausen, examiner
Prof. Kai Nagel, co-examiner
Prof. Hani S. Mahmassani, co-examiner

2008

Contents

List of Figures

List of Tables

Abstract

Our life is strongly influenced by travel which enables the mode of living we are used to in the first place. This importance of travel and the resulting desire for reliable and effective transportation drive the need for predictive models concerning our use of transport infrastructure.

It is remarkable that usually travel is not an end in itself; that is, the motion of traveling is most often not the reason for it to be performed. Nearly always it merely serves the purpose to either transporting a person or goods from one place to another, to enable new activities not available otherwise.

This thesis follows the conviction that travel behavior can only be modeled and conceived on the level of the individual as it is essential to understand the reasoning behind travel. Therefore, the approach under investigation herein is agent based traffic modeling. Agents, as computational representations of real life individuals, act according to simplified rules, by adhering to which they find useful daily activity plans. Then, they execute these plans in a traffic flow microsimulation, producing emerging phenomena such as congestion. The resulting flow patterns and the corresponding agents' daily activity plans can later be used for traffic and travel behavior analysis.

Representing regional travel behavior on the level of individuals creates major challenges in terms of computational costs of such methods. The aim of this thesis is to present algorithms and implementations able to reduce this high demand of resources, rendering agent based models practicable for engineering consultants on standard computing hardware available in most offices today.

Zusammenfassung

Unser Leben, so wie wir es gewohnt sind, wird stark durch Verkehr beeinflusst, Verkehr, der unser Leben überhaupt erst möglich macht. Die Wichtigkeit des Verkehrs und das daraus folgende Bedürfnis nach effektiven Transportsystemen sind es, die den Wunsch nach verlässlichen Vorhersagemodellen für unsere Nutzung der Verkehrsinfrastruktur nähren.

Es stellt sich heraus, dass Verkehr kein Phänomen ist, das sich unabhängig von anderen Einflussfaktoren verstehen lässt. Reisen dienen beinahe immer einem bestimmten Zweck: Einerseits bringen sie uns zum Beispiel an einen Ort, an dem wir eine bestimmte Aktivität ausführen können, die an unserem vorherigen Ort nicht oder nur eingeschränkt verfügbar ist, andererseits ist vielleicht ein bestimmtes Gut an unserem jetzigen Aufenthaltsort nicht verfügbar, wärend es anderswo problemlos erhältlich ist.

Diese Dissertation folgt der Überzeugung, dass Verkehrsverhalten nur auf der Ebene des Individuums verstanden werden kann, da es essentiell ist, dazu die Beweggründe für eine Reise zu erfassen. Aus diesem Grund wird in dieser Arbeit die agentenbasierte Verkehrsmodellierung untersucht. Agenten - das sind die informatischen Entsprechungen zu Individuen im realen Leben - verhalten sich nach vereinfachten, ökonometrischen Gesichtspunkten. Dies führt sie letztendlich zu vernünftigen Tagesplänen. Anschliessend werden diese Pläne in einer Verkehrsflussmikrosimulation ausgeführt, was zu emergenten Phänomenen, wie zum Beispiel der Staubildung, führt. Diese Resultate und die dazugehörigen Tagespläne können anschliessend für Analysezwecke verwendet werden.

Der Wunsch, das Verkehrsverhalten in einer ganzen Region auf Individuen-Ebene abzubilden, stellt eine grosse Herausforderung dar, was den Rechenaufwand solcher Methoden betrifft. Das Ziel dieser Dissertation ist es, Algorithmen und deren Implementationen vorzustellen, welche diesen Rechenaufwand soweit reduzieren, dass die agentenbasierte Verkehrsmodellierung für Ingenieurbüros auf Standardrechnern, die heute in den meisten Büros zur Verfügung stehen, anwendbar wird.

Chapter 1

Introduction

1.1 Motivation

Travel is a very important part of our life. We spend a lot of time traveling, although in most cases we don't like to do so. The reason for traveling is the benefit associated with overcoming space. This enables us, for instance, to perform activities at different places or to transport goods from one spot to another. All this is associated with utility which most often exceeds the various costs associated with travel.

People act mostly individually, without thinking too much about their effect on the rest of the world. Such effects are, for instance, consumption of resources of all kinds, pollution, congestion, price fluctuations, and many more. A beneficial decision for one person or a group may negatively affect other persons in one or another aspect. Overall, such effects may lead to a negative development of some or all parts of our life.

On the other hand, we have an inner need for our life to become better and more comfortable: we want more money, more spare time and more opportunities to use it. So, we have an incentive to try to prevent negative consequences to the world and each individual. Even more, we can try to promote positive effects. To sum up, we want to influence our world to develop in a direction we want it to go, both on the long and the short term.

No matter what the desired development, the question is what are the right actions to take to achieve that goal. It is clear that if we refuse to decide hit or miss, we need an expectation of the effect on the future of a specific measure. This expectation can be seen as the generalization of the concept of prediction. Unfortunately for the modeler, life is complex: everything might have an effect on everything else, and therefore, asking for a prediction (or a predictive model) is much easier than coming up with one. We can, of course, turn to experts and request their view of future development, but the base of such predictions is difficult to formalize apart from the obvious potential of neglecting substantial aspects. Therefore, to get reliable predictions based on explicit assumptions, we need to create models covering all relevant aspects of human life in society.

If we want to make traffic forecasts, we need to know the relevant aspects in life contributing most to travel. For this, one observation is crucial: travel does almost never happen without reason. Instead, the benefit of travel lies in the increase of opportunities achieved. Quite often, it is an activity unavailable otherwise that drives the need for travel. Realizing this, we can say, from the modeler's point of view, we "only" have to create a model of person activities together with the selected location to get a sophisticated travel demand model. Traffic can then be looked at as an emerging phenomenon of such an integrated activity model.

It is a striking fact that traditional aggregated traffic models lack such an explanation of traffic. They merely treat travel demand as a primary and inevitable need that does not react to changes in the world and has no reasoning behind it apart from reproducing statistics.

Many aspects of life influence the activity planning process. There are simple ones like the weather, a persons salary or the fuel price, and there are more complex ones like birth of children, social relationships or moving. In general, the more aspects are reflected in a model the better the predictions potentially are and the wider the field of applications of the model will become. On the other hand, it is clear that we cannot simulate all processes in our model if we want to get results at reasonable costs. Too complex models will need a lot of time and money to be developed, and the required data to run these models will be difficult and expensive to acquire. Also, in such a universal model, many sub-models will have little to no influence on the phenomenon under investigation raising the question if we really need such a model. The bottom line is that we need to find a compromise between refinement of the model and effort to develop it.

Within this work, we hold the view that the most important part of life influencing the short term (up to 3 years) travel demand is activity planning. This includes deciding on activity type and order, selecting the location, choosing the start and end times, deciding on travel modes, and many more. By assigning an expected utility to each activity and using the concept of utility maximization, people select useful activities while taking into account the expected travel costs.

We believe that modeling such complex processes on an aggregated level describing zones as a whole is cumbersome and also difficult to analyze in the end. Instead, we choose the approach of agent based modeling representing every virtual person (the so called agent) individually. This simplifies the model a lot, since we only have to model the individual decisions instead of complex multi-person decisions.

At the end of running such an agent based model, the global resulting patterns are all emerging phenomena arising from the large amount of decisions made by the predominantly uncoordinated individuals modeled.

1.2 Four Step Process

Traditional aggregated modeling, for example, Sheffi (1985); Ortúzar and Willumsen (2001), usually starts by subdividing the region of interest into traffic analysis zones. These zones can be chosen geometrically, geographically, politically, population based or based on any other suitable measure. These zones then need to be connected to the underlying road network using connector links.

In the first step of the process, for each of the zones the number of originating and arriving trips is computed. To do this, sociodemographic data is used. For instance, the number of residents in a zone together with employment data might be used to estimate the number of originating work trips from that zone. Similarly, the number of jobs in a zone can be used to estimate the number of arriving work trips. Using similar ideas for other trip purposes, the total amount of originating and arriving trips can be calculated. This step is normally referred to as *trip generation*.

From this point on, it is fixed how many trips will be starting and ending in each individual zone. However, this does not make any statement on where the trips are going or where they are coming from. Finding an answer to this question is the objective of the second step of the four step process, *trip distribution*. Assuming an equal total amount of originating and arriving trip, for each origin destination pair the number of trips has to be computed. Most often a *gravity model* is used to provide the missing information. Gravity models make the assumption that the probability of using a specific origin destination pair is depending solely on the number of originating and arriving trips in the respective traffic analysis zones and the (average) costs to travel from the originating zone to the destination zone. Basic gravity models do not necessarily reproduce the desired totals of originating and arriving trips. If reproduction of totals is desired—and most often it is—the gravity model has to be constrained on both sides by use of an iterative process. The result of trip distribution is called the origin destination matrix (or shortly OD-matrix) which contains for each origin-destination pair the number of executed trips.

Having the totals of the trips, the next question addressed is the *mode choice* per origin destination pair. There is a wide range of ways to solve the mode choice problem from very simple to more sophisticated ones. One common approach is based on the estimated (average) costs of travel per origin destination pair using the different modes at hand. Additionally, general preferences and quality of service assumptions may be used. All this is then used to run some sort of discrete choice model to find the desired percentages per mode. If the real modal split is known from survey data it is often required to reproduce this specific number, then, additional measures have to be taken.

The last step is the iterative *assignment* process. Here, for each origin destination pair, the trips are distributed along all possible routes connecting the corresponding zones. Most often, the basic requirement is to reach a user equilibrium. For each OD-pair this means that all routes from a zone A to a zone B that are loaded with traffic need to realize the same travel costs (see for example, Wardrop, 1952). The travel costs on the route are approximated by a sum of continuous functions that use link capacity and link load as input and return the corresponding link travel time. To find the correct link loads, an iterative optimization process is run. The resulting loads on each link can either be used directly for traffic analysis and planning, or indirectly by converting the loads into equivalent estimated travel times and travel costs per origin destination pair.

1.2.1 Limitations of the Four Step Process

There are a number of limitations or challenges when using the four step process for transport planning. Most of them emerge from the basic design of the approach, others are less fundamental and rather represent a problem of specific implementations.

- Memory demand with increasing resolution: When using a traffic model with N traffic analysis zones, the origin destination matrix will have $N \times N$ cells. During the assignment step, each of these cells has to store a multitude of routes connecting the zones involved. With increasing resolution and area of interest, the number of zones will go up very quickly as will the average route length and number of alternative routes between two cells. This imposes severe limitations to how well the approach scales to larger problem sizes.

- Attachment of zones to network: The zones need to be attached to the modeled road network by using so called connector links. The placement of these connector links defines where the travel demand enters the network and can strongly influence the outcome of the computation. In general, connecting the zones to the road network requires expert knowledge, is time consuming and costly, and has similar scaling problems as the OD-matrix.

- Trip-based structure: The whole approach described above is essentially trip based. There are no activities involved and therefore no activity chains can be observed.

- Missing explanation of trips: For the same reason, trips in the model lack any explanation of why they take place.

- No utility of trips: The reason for this lack of explanation is that there is no utility assigned to the trip (or rather the activity requiring the trip). Furthermore, this utility is important for many analyses when trying to understand travel behavior.

- No feedback between steps: Each of the four steps stands for itself and there is no feedback from a later step to the earlier ones. This missing feedback is a very strong simplification of reality. It even presents a laxity in the definition of the four step process, as in fact, both, trip distribution and mode choice need an estimation of travel times between zones which is not available until the traffic was assigned to the network in step four. This gap is often filled with some other estimate of the travel times, for instance, the travel distance, but this cannot be regarded as a good solution.

- Static nature: The described process only handles overall volumes without any notion of time of day. As a result, many interesting questions involving temporal patterns cannot be answered using this method.

- Fixed demand: As travel demand is derived solely from sociodemographic factors, it cannot be used to predict changes in demand based on a change in the road infrastructure. Especially the number of trips will stay the same as long as the base data for the trip generation process remains the same.

Some of the described limitations can be overcome by extending the presented model. For instance, there are extensions to get hold of the dynamic nature of traffic. Basically, this is done by running similar processes as described above for every hour of the day and by linking these hourly results accordingly. Unfortunately, with increasing resolution, this approach makes the memory problem even worse. Limitations of the trip generation step can be alleviated by using so called activity based models. Further on, successive re-execution of some of the steps using results from previous iterations is a relatively simple method to solve or at least lessen the missing feedback problem.

However, there are other limitations more difficult to deal with, such as, missing activity chains in the final assignment, the above described memory demand, and the prediction of change in total demand, based on road infrastructure changes.

1.3 Agent Based Transport Modeling

Agent based transport models follow a very different approach to transport modeling by staying much closer to the individual and having representations for all major parts of daily life that involve travel in some way. Conceptually, each individual is represented by a virtual *agent*. All agents together form a synthetic population in a virtual world. See for instance, Nagel and Barrett (1997); Marchal (2001); Raney (2005); Balmer (2007). Apart from the agents, this virtual world contains representations of infrastructure, such as, the road network and facilities like houses, working places, shopping malls, and recreational areas. Each agent has a plan taking place in the virtual world. This plan is created by the agent himself[1] based on rational assumptions.

When planning is done, execution starts which is where the virtual world comes to life. During execution, agents interact while using the infrastructure, and the system's response is what really happens during the day, as opposed to what the agents planned. Common examples of emerging phenomena through joint use of infrastructure are congested roads or overcrowded malls and parks.

The plans and the system response can be quite different especially when the assumptions made during the planning proved to be far off. In this case, the affected agents need another chance to bring their plans closer to the virtual reality. To provide such a mechanism, an integral part of the model is iteration, where the results of the last execution are available as base for the replanning process. After convergence, the plans contain the final travel demand, and the response of a fully developed agent based transport model contains information about each agent's detailed trajectory during the day including activities and travel modes. By post processing this data, aggregated information can easily be extracted, like hourly traffic volumes and time dependent travel times for each link or origin destination pair. But note: the result of an agent based transport model contains much more than just traffic volumes on links. Rather, something similar to very detailed person diaries (of virtual agents) is generated.

[1] Throughout this dissertation, the female form (e.g., she, her, herself, etc.) is implicitly included in the male form.

Four step process	Agent based transport model
Trip generation	Activity pattern generation + location choice
Trip distribution	Location choice for activities
Mode choice	Mode choice per activity pair
Traffic assignment	Routing + plan execution + replanning

Table 1.1: Comparison of the four step process with agent based transport models.

1.4 Comparison of Four Step Models and Agent Based Transport Models

Let us compare the two approaches to transport modeling to see how different parts take similar roles in the final methods. See Table 1.1 for a comparison summary of the two models.

It can be seen that basically all parts of the four step process have an equivalent counterpart in an agent based model: when an agent decides to change his daily plans activity pattern, especially if a new activity is added, and if the assigned locations differ from the preceding and the subsequent activities, a new trip has to be carried out. This can be interpreted as a trip being generated. Similarly, the location choice process as a whole is carrying out trip distribution functionality. Here again, the net effect of location choice on trips depends on the locations. For subsequent activities at the same location no trip is carried out, otherwise a trip is created and at the same time “distributed”. In agent based traffic models, mode choice can be carried out similarly to the four step process, that is, for each pair of subsequent activities a travel mode can be selected. However, care has to be taken not to create an illegal daily plan. Many mode patterns are not feasible, as the mobility tool for each mode might not be available at all locations. For instance, if the home to work trip was carried out by train the opposed trip cannot be accomplished by car, since the needed vehicle is still parked at home and not at the work location. This is not a limitation of agent based models, however. Rather, with traditional approaches, the production of such illegal mode patterns can easily stay unnoticed, which is not a desirable situation. Finally, the assignment step finds its counterpart jointly in the daily planning and in the traffic flow microsimulation: finding the routes is covered by the agents, and computing travel times for these routes can only be done by simulating the day.

1.4.1 Parts of the Agent Based System Studied in this Thesis

One important part of an agent based transport model addressed in this thesis is the replanning process, as it represents the virtual counterpart of the real human decision making. The work presented here makes the fundamental assumption that all human acting is based on reasoning according to econometric aspects. That is, we assume that there is a utility (or disutility) assigned to every action an agent can take, and the goal of each agent is to maximize his own overall utility. While this is certainly a simplifying assumption, it also forms a sound base for this work, as it turns the very complex human decision making into a well defined optimization problem. This optimization has to take place in “planning space” which is a high dimensional space consisting of discrete and continuous dimensions. The utility assigned to actions and the resulting optimization problem is addressed in Chapter 2.

The described approach treats daily plans as unchangeable during the day. While this is an assumption useful for finding an equilibrium state of agents routinely performing similar activities over and over again, it is wrong if we look at responses to unforeseen events. One way of approaching this type of reaction is investigated in Chapter 3.

From the point of view of the overall system, agent learning needs to be coordinated, as otherwise all agents adapt their plans in each iteration leading to overshooting effects, and the system would not be able to converge to any point of rest. In the simplest version, random selection of a certain fixed amount of replanning agents per iteration will solve the problem and make the system operational. However, Chapter 4 shows that the overall performance of the system can be greatly improved by an optimized learning strategy.

Plan execution creates the link between plans and virtual reality and therefore is one of the most important parts of the overall system. The traffic flow microsimulation takes on this job, and very often, it represents the most expensive part of the whole system in terms of computational effort. How this can be alleviated and optimized by simulating discrete travel events and through parallelization, as well as how the dynamics of the microsimulation can be improved is discussed in Chapters 5–8.

1.5 Structure of This Dissertation

The largest part of this thesis is formed by papers published in journals or presented at conferences. These papers were newly typeset to fit into the formatting of the presented work.

In Chapter 2, we demonstrate an approach to agent based daily planning restating the task as an optimization problem of daily utility. A utility function of activity execution is defined, and the final optimization is carried out using a genetic algorithm (GA). GAs are population based stochastic optimization algorithms from the field of evolutionary computation. Borrowing from nature, possible solutions to the optimization problem at hand are encoded in a genome which is then modified using mutation and crossover operators. From the resulting new candidate solutions the bad ones are sorted out by a selection step, thereby driving the set of candidates towards an optimum. The final program is tested for performance using selected artificial test problems which include optimal learning of very busy and rather quiet days.

Chapter 3 works on an extension to daily replanning: within day replanning. The approach followed is to create a more extensive daily plan, namely a plan including the best reactions to all possible delays during the day. By formulating this task as a reinforcement learning problem, it can be solved by the Q-learning method from the field of artificial intelligence. By using a balance between exploring the space of possible states and exploiting knowledge acquired before, this method can find globally optimal solutions to the problem at hand. Computational results complete this chapter.

Replanning is again the focus of Chapter 4. First, the time allocation part of daily replanning is approached using the covariance matrix adaptation evolution strategy (CMA-ES). CMA-ES has shown to perform well on a variety of optimization problems including such with noisy and distorted search spaces. This efficiency makes it possible to use the real time-dependent travel times in the replanning step. Second, the surrounding learning strategy, namely the percentage of replanning agents per iteration, is investigated. We show that a decreasing percentage can improve the overall learning speed significantly, cutting down the number of iterations needed by at least a factor of five.

A new microscopic queue-based traffic flow simulation is presented in Chapter 5. Extending the traffic flow model already used in our group by introducing the notion of backwards traveling gaps, it uses the discrete events of interest (the times when cars enter or exit links) to actually drive the simulation forward in time. By doing so, it eliminates inefficiencies present during off-peak times and on very congested roads. This makes it possible to run large traffic flow microsimulations with more than one million agents on planning networks in less than ten minutes on single-CPU desktop computers.

Chapter 6 goes even further and shows how the same model can be parallelized to run on 64 processors or more. This is done by appropriate load balancing and by minimizing interfaces between processors. These small interfaces lead to reduced communication needs. The final parallel microsimulation, using a shared memory machine with 64 CPU-cores, can simulate a 24 hours scenario with 7 million agents on a 28k links network in 87 seconds.

The properties of our extended traffic flow model are investigated in Chapter 7 using a ring test network and stochastic demand. The resulting flow-density-diagram has a trapezoidal form and can be described as a stylized form of fundamental diagrams known from real world measurements.

Using queue-based microsimulations for urban traffic can be difficult since signaled intersections have a big impact on observable flow patterns. To alleviate this problem, we extend our traffic flow model by time dependent green time fractions in Chapter 8. The idea is that the main use of traffic lights—apart from regulating intersection use—is mainly to adjust the available flow capacity on intersecting roads. Exactly this, the percentage of green time per link, can now be controlled directly using the presented extension of our microsimulation model.

Finally, in Chapter 9, the results of the presented work are summarized, and Chapter 10 gives an outlook on future work.

Chapter 2

Generating Complete All-Day Activity Plans with Genetic Algorithms

Authors

David Charypar
Institute for Transport Planning and Systems, ETH Zurich, Switzerland
Email: charypar@ivt.baug.ethz.ch

Kai Nagel
Institute for Land and Sea Transport Systems, TU Berlin, Germany
Email: nagel@vsp.tu-berlin.de

Publication

This paper was presented at a conference (Charypar and Nagel, 2003) and later published in a journal (Charypar and Nagel, 2005).

Abstract

Activity-based demand generation constructs complete all-day activity plans for each member of a population, and derives transportation demand from the fact that consecutive activities at different locations need to be connected by travel. Besides many other advantages, activity-based demand generation also fits well into the paradigm of multi-agent simulation, where each traveler is kept as an individual throughout the whole modeling process.

In this paper, we present a new approach to the problem, which uses genetic algorithms (GA). Our GA keeps, for each member of the population, several instances of possible all-day activity plans in memory. Those plans are modified by mutation and crossover, while "bad" instances are eventually discarded.

Any GA needs a fitness function to evaluate the performance of each instance. For all-day activity plans, it makes sense to use a utility function to obtain such a fitness. In consequence, a significant part of the paper is spent discussing such a utility function. In addition, the paper shows the performance of the algorithm on a few selected problems, including very busy and rather non-busy days.

Keywords: Activity generation; Utility functions; Genetic Algorithms; Location choice; Multi-agent traffic simulation

2.1 Introduction

The larger context of the work presented in this paper is the attempt to build an integrated multi-agent simulation model for transportation planning, "multi-agent" meaning that each traveler in the simulation is individually resolved. Multi-agent simulations can be employed on many levels, from housing choice down to driving behavior. Our own initial goal is to replace the four-step process by a multi-agent simulation. This implies the following modules and methods:

- The process starts by generating a *synthetic population* from census data (e.g. Beckman *et al.*, 1996).
- Next, for each synthetic person of the synthetic population a *plan* is generated. Plans consist of activity patterns, activity locations, activity times, mode choice, route, etc. (e.g. TRANSIMS, 2006; Pendyala, 2004; Bhat *et al.*, 2004).
- Up to here, the computations of the agents are essentially independent, apart from possible small-scale coordination problems such as household coordination or ride sharing. In contrast, in the *mobility simulation*, all agents' plans are simultaneously executed and the results of interaction are computed (e.g. MATSim-T, 2004; DYNASMART, 2003). One important interaction result is congestion. – Note that most traffic micro-simulations do not truly execute agent plans at the route level, but rather keep the travelers' destinations and have the routing done by the network.
- As is well known, the causal relation between the modules goes into both directions. For example, if many agents choose activities at many different and far apart locations, then this will cause congestion. This congestion will cause them to select activities which necessitate less travel. The typical way to solve this problem is to use *iterations* between the modules (e.g. Cascetta, 1989; Kaufman *et al.*, 1991; Nagel and Barrett, 1997). This can be either interpreted as relaxation or as human learning.

For this approach, many collaborating modules need to be designed, implemented, and tested. An important part of those modules concerns activity generation: for each synthetic individual, a sequence of activities is generated, including activity location and activity times. Activity-based demand generation is a very active field of research. The mainstay of activity-based demand generation are random utility models (RUMs) (Ben-Akiva and Lerman, 1985; Bowman, 1998; Pendyala, 2004; Bhat *et al.*, 2004). RUMs, however, arguably have the disadvantage that they are behaviorally not very realistic. In consequence, alternatives are also investigated, such as behaviorally or rule-based approaches (e.g. Arentze *et al.*, 2000; Miller and Roorda, 2003).

The question that will be considered in this paper is in how far Genetic Algorithms (GA) can contribute to the field of activity generation. GA are biologically inspired optimization methods that are relatively inefficient computationally but extremely flexible. In consequence, the question to be treated in this paper is if this flexibility can be stretched to include activity generation, and what the resulting computational burden is.

The paper starts with a more precise problem description (Section 2.2), followed by a short review of previous work in the area of activity generation (Section 2.3). Section 2.4 then discusses a concept of how GA could be used to generate daily activities; Section 2.5 contains details about our specific computational implementation. GAs work by maintaining a population of solutions; they improve the best known solution by mutating and combining members of that population. In order for this to work, individuals need to be given scores. This is normally called a fitness function; in social science research, it is plausible to use utility functions instead. In consequence, Section 2.6 describes the requirements that a utility function for the GA approach needs to fulfill, and which particular utility function we selected. However, *any* function that gives scores to activity chains will work. Section 2.7 then contains tests and results for several illustrative examples. The paper is concluded by an outlook on future work (Section 2.8) and a summary (Section 2.9).

2.2 Problem Description

The problem of activity planning is the task to generate a complete activity plan for an agent from a set of possible activities (an activity repertoire). A complete activity plan stores which activities are to be executed and in which order, it assigns a location to each activity and also an execution time and a duration.

Activity planning divides into three subproblems:

- The first subproblem is to select activities to be executed and to decide in which order they should be executed. We refer to this subproblem as *activity pattern generation*.

- The second subproblem is to find the places where the activities are going to be executed. We call this *location selection*. The location selection has to fulfill a number of constraints in order to be meaningful. For instance children should be fetched from school at the same place where they were dropped off before.

- The third subproblem is to decide when the activities have to be executed and for how long. We call this *time allocation*. Once again, time allocation is subject to several restrictions. The most simple one is that the execution times should be ordered in correspondence with the activity pattern.

In reality, all of us do activity planning every day with sufficient speed and satisfactory results. Nevertheless this problem is quite difficult to solve automatically on the computer. The main problem with activity planning is the huge amount of possible plans for a given set of activities. Even if one is just concentrating on *activity pattern generation* for a list of ten activities, there are almost 10 million possible solutions. It is clear that we get even more problems if we want to include *location selection* and *time allocation*.

To make the problem even worse, it would be desirable to extend the length of the activity plan to a week or even a month because day plans are not independent in general as some activities do not have to be executed every day. For instance you do not have to go shopping every day, but you also cannot omit shopping for a longer period.

Since the space for possible solutions scales exponentially with the length of the desired activity plan, generating complete week plans is a problem that is several orders of magnitude more complex than generating day plans.

2.3 Related Work

There are many models tackling the same problem. One can maybe differentiate the following directions:

(1) A possible way to solve this problem is to use a nested multinomial logit model. One such system is described by Bowman (1998). The decision is decomposed into many hierarchical levels, such as: the choice between different activity patterns, the choice between different locations, the choice between different starting times for the patterns, etc. This method demands that approximations of the lower level results are available at the upper levels: for example, in order to decide between different activity patterns, it is necessary to have a performance estimate for each pattern, which can only be obtained if the algorithm has some idea about the locations and times that will be chosen for each pattern. In practice, this is achieved via the so-called logsum terms, which back-propagate the lower level solutions to the higher levels. That is, the algorithm starts at the leaves of the decision tree. There, it computes, for each given activity pattern and location choice, utilities for each possible time choice. It then calculates the expected utility from this, and passes this on to the location choice level. The location choice level then calculates, for each given pattern, the expected utility for each location choice and passes the resulting expected utility for each pattern one level up, etc. Once the algorithm is at the highest level, it selects between the patterns according to the utilities. Once the pattern is selected, for this given pattern it selects between the locations. Once the locations are selected, it decides on the time-of-day when the pattern is started.

Discrete choice models have a similar conceptual approach as our model in that they make choices based on utilities. The two main differences are that, at least conceptually, discrete choice models enumerate *all* possible alternatives, and that they do not choose the option with the best utility but they choose between options with probabilities which are related to utilities. The first aspect means that a huge number of options needs to be considered; the second (behaviorally somewhat justified) aspect means that efficient search methods such as branch-and-bound cannot be used because even “bad” branches of the search tree have a probability that they will be selected.

(2) An arguably related approach is STARCHILD and successors. Instead of making a probabilistic choice between different options, it finds the optimal solution. Because of this simplification, methods from mathematical programming can be applied (e.g. Recker, 1995).

(3) Jara-Diaz and Guerra (2003) look at complete daily schedules in terms of an econometric interpretation. For example, the ratio of the durations of work versus non-work is explained by a combination of the value of time and the wage rate.

(4) All approaches mentioned so far always look at the complete schedule. As an alternative, a traveler may build the schedule as he/she goes. An extreme version of this is PCATS (e.g. Kitamura, 1996), which conditions decisions on the past history, but does not look into the future. The advantage is much more manageable computational complexity; the disadvantage is that the algorithm does not pick up scheduling constraints which lie in the future.

(5) The work by Doherty and coworkers (e.g. Doherty and Axhausen, 1998) implies that real-world activity scheduling is a combination of the above aspects, i.e. that some decisions are made a long time in advance while others are rather spontaneous. An implementation of this approach is ALBATROSS (e.g. Arentze *et al.*, 2000).

(6) Miller has attempted to build a model that is considerably more process-oriented than typical RUM models. It was applied to the Toronto metropolitan area (Miller and Roorda, 2003).

2.4 Idea: Genetic Algorithms

Trying to solve the problem by enumerating all possibilities—a complete search—is infeasible. This is especially true if one has only very limited computer time for program execution, as is the case with large scale multi-agent implementations. Furthermore, for our problem it is not absolutely necessary to find the global optimal solution. In fact, in many cases what people use as their plan is far from being optimal. It would be sufficient to find a “good” solution.

The idea for this paper is to use a Genetic Algorithm (GA) to find good all-day activity plans. GAs have been used for many problems with huge search spaces; a suggestion to use them in the context of transport/land-use research is by Abraham and Hunt (2002). GAs maintain a population of solution instances during the search process, and search progress is made by mutation, crossover, and selection, as explained below. In our case, the population of solution instances consists of many possible day plans for a single given traveler. The quality of such a day plan is rated by a fitness function which uses information and restrictions known about the activities, and estimates how well the day plan meets them.

A random population of such day plans is created at the beginning of the algorithm. New individuals are created by crossover between two good day plans (the "parents") and mutating the offspring.[1] When a better day plan is found, then the worst plan is removed, keeping a population of constant size. This procedure is repeated a number of times. At the end, the best day plan is used as solution of the problem.

Note that "population" of "individuals" is used with two different meanings in this text: first, there is a population of travelers that populates the multi-agent traffic simulation, but second, there is also a population of solution instances within the GA. In the remainder of the text, the second meaning will be used if not stated otherwise.

2.5 Implementation

As mentioned above, crossover, mutation and selection are vital components of each GA. Before these operators can be defined, it is necessary to come up with a way of representing a solution instance in the computer. This way of representation is referred to as *encoding*. Encoding is of prime importance for the definition of crossover and mutation and it has a large influence on the potential performance of a GA.

For our problem of activity planning, we used a combination of binary encoding, permutation encoding and value encoding in the following way (Figure 2.1):

[1] In this paper, the word "parents" is only used in the computational sense, never in the biological sense.

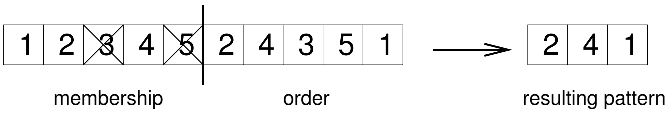


Figure 2.1: Encoding of activity pattern "241"

- Each activity (of a given and fixed set) can be either included in the day plan or excluded from it (binary encoding). This information—which we will refer to as membership information later on—is stored in an array of bits, where each bit holds this information for one activity. If a bit is set to one the corresponding activity is included in the day plan (i.e. it is a member of the day plan), if it is set to zero the corresponding activity is excluded from the plan.

- A second array stores the order of the activities (permutation encoding). It holds always a full set of activities, regardless of which activities are actually member of the day plan and which not. When evaluating the day plan, the positions with disabled activities are ignored.

- Our day plans also include information about the location choice. In our model we assume that there exists a facility that is assigned to each activity. This facility can be thought of as a type of building or place that is needed in order to perform the activity. For instance, for the activity "go shopping" one needs the facility "shop". We also assume that for each facility there exist multiple locations as for example there are multiple shops in a city where shopping can be done. The concept of facilities makes it possible to have locations of activities that depend on each other. For instance, this is needed to take care of the fact that one has to fetch his children at the same school as one has dropped them off before. The day plan has to store the selected location for each facility. This information is held in a third array storing the ID of the location for each facility (value encoding).

- The activity durations are stored in a fourth array which stores floating point numbers (once more value encoding); in addition,

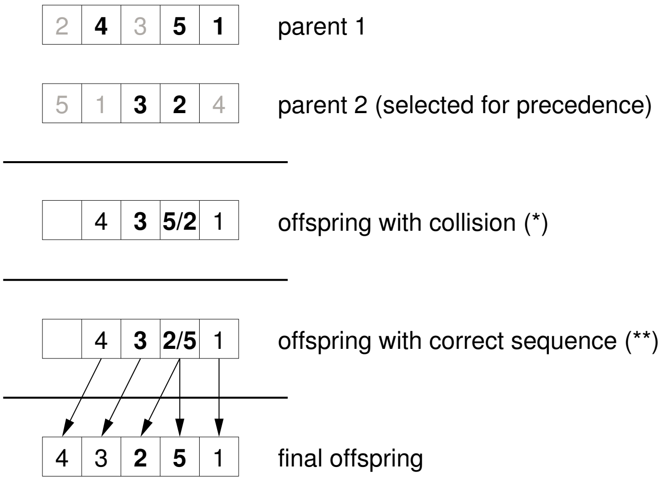

2 4 3 5 1
parent 1
5 1 3 2 4
parent 2 (selected for precedence)
4 3 5/2 1
offspring with collision (*)
4 3 2/5 1
offspring with correct sequence (**)
4 3 2 5 1
final offspring

Figure 2.2: Illustration of the crossover operator

there is an entry which contains the starting time of the day plan. Activity starting times of the allocated time slots are a result of adding up all previous durations and travel times (see below).

The parents for a new individual are selected at random out of the current population. When the offspring is better than the currently worst member of the population, then the worst member is replaced by the new offspring. Otherwise, the offspring is not kept. Since there is no selection at the parent level, all existing solutions except the worst are treated equivalently, which maintains a relatively large degree of diversity in the population. Slow progress towards better solutions is made by removing the worst member.

The implementation of the **crossover operator** is built of three parts:

- First, the array which stores the membership information is processed using uniform crossover. That is, each membership bit is copied randomly from either one of the parents.

- Second, we crossover the arrays that store the order information (Figure 2.2). Before we start, we decide randomly which parent should precede (i.e. which parent's activity should come first) in case of a collision. For each activity ("1" to "5" in Figure 2.2), we randomly select a parent and put the activity at the same position in the offspring as it was in the selected parent (see "(*)" in

Figure 2.2). If there are collisions (i.e. two activities in the same cell), then first, their sequence is decided (see "(**)"), and then, the resulting new sequence is copied into its correct location. This somewhat involved algorithm was developed since it does not depend on the sequence in which it goes through the activities, and therefore it does not introduce a bias caused by the ordering in which the activities are processed in this sub-step.

- Third, for each activity, one of the the two parents is randomly selected, and the new duration is set to the duration of the same activity in the selected parent. Note, that this makes it quite possible that the position of an activity in the sequence order is taken from one parent, but the duration of the activity from the other parent.

Note that standard single point crossover would not work. In single point crossover, the offspring is created from the parents by taking their representations, choosing a random crossover point and copying the first part up to this point from the first parent and the second part beginning at this point from the second parent. However, for sequencing problems such as ours, this approach does not work since in general, in the offspring some activities will be performed twice while some others will have vanished.

A solution, coming from GA encoding for the Traveling Salesman Problem, is to take the first part verbatim from the first parent, and then to fill up the remaining spots with the remaining activities in the sequence they are in the second parent, skipping activities which are already present in order to avoid duplicate activities in the offspring. That encoding has, however, the disadvantage that it only looks at the sequence and not at all at time-of-day. In contrast, in our implementation, activities have a tendency to stay at their position within the day plan. In addition, our crossover operator shuffles the activities more than a single point crossover would, and our tests showed that this yields a more stable—although slower—convergence. This is desirable, because our experience with GAs shows a considerable tendency to keep stuck in local optima when using single point crossover.

The **mutation operator** is split into four parts: First, for each activity, the membership information is flipped with a probability p_{mut}. Second, the order of the activities is permuted: With a probability p_{mut} two activities, both chosen at random, are exchanged. This is done n

times where n is the total number of activities. Using the same idea as in the crossover operator, the activities are exchanged ignoring the membership information. Third, the duration of each activity is changed by multiplying it with a factor $f = e^X$, where X is drawn uniformly from the interval $[-p_{\mathrm{mut}}/2, p_{\mathrm{mut}}/2]$. Fourth, a new start time of the activity plan is calculated by adding a random time uniformly drawn from $[-p_{\mathrm{mut}} \cdot 12\mathrm{h}, p_{\mathrm{mut}} \cdot 12\mathrm{h}]$.

2.6 Utility Function

As already mentioned earlier, our GA needs to rate the quality of the day plans in the population. For that purpose one has to define a *fitness function* which somehow defines what a “good” day plan is. An important advantage of using a GA is that the fitness function can be changed very easily according to the preferences of the user.

We use a fitness function that is the sum of the utilities of all activities that are performed, plus the sum of all travel (dis)utilities:

$$F = \sum_{i=1}^{n} U_{\mathrm{act}}(type_i, start_i, dur_i) + \sum_{i=2}^{n} U_{\mathrm{trav}}(loc_{i-1}, loc_i) \qquad (2.1)$$

Here, $type_i$ is the type of the activity, $start_i$ is the starting time of the activity, dur_i is the amount of time allocated to the activity, and loc_i is the location of the activity. In the following part, we will discuss all aspects of our utility function in detail.

In our model, the utility of an activity depends on the following variables:

- The time of day when the time slot for the activity starts
- The allocated time to the activity
- The location where the activity takes place
- The location where the last activity took place.

The utility of an activity i is—in our model—the sum of four terms, each of which is modeling a certain aspect of the utility function.

$$U_{\mathrm{act},i} = U_{\mathrm{dur},i} + U_{\mathrm{wait},i} + U_{\mathrm{late.ar},i} + U_{\mathrm{early.dp},i} + U_{\mathrm{short.dur},i} \; . \tag{2.2}$$

$U_{\mathrm{dur},i}$ denotes the utility of executing the activity for a certain duration, $U_{\mathrm{wait},i}$ denotes the (dis)utility of waiting (for instance waiting for a shop to open), $U_{\mathrm{late.ar},i}$ and $U_{\mathrm{early.dp},i}$ denote penalties for coming too late or leaving too early, respectively, and $U_{\mathrm{short.dur},i}$ is a penalty if an activity is performed for too short a time.

U_{trav} denotes the (dis)utility of traveling from the last location to the next one.

This approach has the consequence that when removing an activity, the travel terms at *both* ends are modified. That is, if an activity is far out of the way, then dropping that activity will reduce overall travel considerably, while dropping an activity that is on the way will have a negligible effect on travel times.

In the following, the different terms and their parameters will be discussed in detail. All terms except U_{dur} are modeled to be linear in the time needed for that activity. Despite the detailed discussion, it should be kept in mind that the technology of using a GA is entirely independent from the specific utility function. If a different utility (or general scoring) function is desired, it is very simple to replace it.

2.6.1 Utilities for Performing Activities

Although fitness functions can be easily replaced in GA approaches, a specific fitness function needs to be selected in order to run tests. We decided to use a logarithmic function as utility of duration:

$$U_{\mathrm{dur}}(t_{\mathrm{dur}}) = \beta_{\mathrm{dur}} \cdot t^* \cdot \ln(\frac{t_{\mathrm{dur}}}{t_0}) \; . \tag{2.3}$$

Here, t_{dur} is the duration of the activity as it is actually performed, and β_{dur}, t^*, and t_0 are parameters, to be explained later.

Logarithmic utility functions where introduced by Bernoulli (1738) as a solution to the St. Petersburg paradox. They have the property that the marginal utility of doing more of the same activity is decreasing with longer durations, but it is always positive. This may seem implausible because then there is nothing that limits the execution time of activities. However, considering more than one activity and a limited time budget, the available time will be distributed in order to achieve a higher overall utility, thus limiting the time spent at each individual activity.

In the absence of other restrictions such opening times, the optimal time allocation for an activity pattern is reached if all activities have the same marginal utility of duration. Otherwise, the agent could gain by reallocating time from activities with small marginal utilities to activities with large marginal utilities.

t^* is the duration at which the marginal utility is β_{dur}, as can be seen by taking the partial derivative:

$$\frac{\partial U}{\partial t_{\text{dur}}}(t_{\text{dur}}) = \frac{\beta_{\text{dur}} \cdot t^*}{t_{\text{dur}}} \,, \tag{2.4}$$

and setting $t_{\text{dur}} = t^*$ yields indeed β_{dur} for the marginal utility. t_i^* gives the typical duration of activity i. Or more precisely: the t_i^* yield the *ratios* of the durations of different activities in equilibrium.

t_0 is the duration at which the utility starts to be positive. It plays a double role:

- It determines the minimum duration of an activity. If the duration falls below this value, then it is more beneficial to drop the activity and do nothing instead.

- It determines the priority of an activity: The marginal utility at $t_{\text{dur}} = t_0$ is

  $$\frac{\partial U}{\partial t_{\text{dur}}}(t_0) = \frac{\beta_{\text{dur}} \cdot t^*}{t_0} \,. \tag{2.5}$$

  If one sets t_0 proportional to t^*, i.e. $t_0 = \alpha\, t^*$, then the proportionality factor α will decide over the marginal utility at t_0 and therefore over the priority with which an activity is maintained when time gets tight.

In our case, the specific form of

$$\alpha = e^{-200€/(t^* \cdot p \cdot \beta_{dur})} \tag{2.6}$$

was used, where p is the priority. This specific form has the consequence that all activities of the same priority have the same utility at $t_{dur} = t^*$.

Note that even activities with a high priority can be dropped if they are very inconvenient. As we will see later, this has sometimes implausible results, such as picking up a child from kindergarten but never dropping it off. On the other hand, it is certainly true that schedules can become so tight that even high priority items are dropped, for example by asking someone else to help out. For that reason, making high priority activities completely obligatory seems not plausible.

2.6.2 Penalties

All penalty terms follow the penalty terms of the Vickrey model of departure time choice (e.g. Arnott *et al.*, 1993) in that they are modeled to be linear in their time consumption:

$$U_{\text{trav}}(t_{\text{trav}}) = \beta_{\text{trav}} \cdot t_{\text{trav}} \ ,$$

$$U_{\text{wait}}(t_{\text{wait}}) = \beta_{\text{wait}} \cdot t_{\text{wait}} \ ,$$

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

(where t_{start} is the starting time of the activity and $t_{\text{latest.ar}}$ is the latest possible starting time for the activity),

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

(where t_{end} is the ending time of the activity and $t_{\text{earliest.dp}}$ is the earliest possible ending time for the activity), and

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

(where $t_{\mathrm{short.dur}}$ is the shortest duration for the activity),

At this point, one could discuss plausible relations between the different β, for example using the typical Vickrey scenario values of $-6€/h$, $-12€/h$, and $-18€/h$ for β_{wait}, β_{trav}, and $\beta_{\mathrm{late.ar}}$, respectively. However, as will be discussed in Section 2.8, in our framework there are interactions between the marginal utility of doing activities and the *effective* marginal utility of the penalty terms, so that the effective marginal utilities that guide behavior are more complicated than one would assume by the above numbers. This is caused by the fact that some of the penalized items, such as waiting and traveling, also incur the *additional* penalty of not being able to earn positive utility from doing an activity at the same time (opportunity cost). The other penalized items—arriving late, leaving early, or staying for too short—do not have this property. Therefore, the values used in the tests will just be stated in Section 2.6.4 and be used without further comment except for the discussion in Section 2.8. – Once more, please note that this paper describes a prototype implementation rather than an operational model.

2.6.3 Opening Times and Similar Constraints

Many activities can only be carried out during certain times of the day. For instance shopping can only be done when the stores are open. The question is what utility we want to assign to activities which violate these constraints.

One possibility would be to assign a very low utility to those activities, e.g. minus infinity. This policy would be very efficient in avoiding invalid day plans. But it would not make very much sense, for the following reason: Assume that you have to compare two time allocations for the activity "shop". In the first scenario you go shopping at 7:59am and shop for two hours. In the second scenario you go shopping at 8:00am and shop also for two hours. The shop opens at 8am. The first time allocation is invalid because you try to shop while the shop is closed. So the utility of the first scenario would be very low. The second scenario, however, has a very high utility. With this policy, the minimal change of one minute in time allocation would produce a huge change in utility which is certainly not realistic. In reality, we would have waited one minute in front of the shop. That means that one should set parameters such that both utilities are almost the same.

Based on these reflections, we come up with the following constraint handling policy:

- At the beginning of the allocated time slot, we assume that the agent travels from the location of the last activity to the location of the new activity. The time spent for traveling yields a (dis)utility according to the section about travel costs.

- From the moment of arrival at the activity location until the end of the time slot, as much time as possible is spent actually performing the activity.

- If for some reason, for instance because of opening hours, it is not possible to use all this time for performing the activity, the remaining time is spent waiting. Waiting yields a negative utility of $\beta_{\text{wait}} \cdot t_{\text{wait}}$.

- Since we use a logarithmic function for the utility of duration, it is possible that this utility becomes negative. If this is the case, and if it is more efficient to spent the whole time waiting[2], the activity is not executed at all. It may sound weird to travel to an activity that is not executed. However, it is important to note that we need to calculate meaningful utilities for no matter which day plans the GA generates, because this is the material the GA works with. Since traveling to an activity location without executing the activity does not make sense, the GA will eventually find a better solution, such as either completely dropping the activity, or allocating sufficient time for it.

2.6.4 Summary of Parameters

The following is a listing of all parameters of our utility function and the values that were used.

[2] The second if-condition is only relevant if β_{wait} is different from zero. As will be discussed in Section 2.8, it makes much sense to set it to zero, in which case this condition is no longer relevant.

Marginal utility of any activity(β_{dur}):	$20\text{€}/\mathrm{h}$
Marginal utility of travel time(β_{trav}):	$-12\text{€}/\mathrm{h}$
Marginal utility of waiting(β_{wait}):	$-6\text{€}/\mathrm{h}$
Marginal utility of coming late($\beta_{\mathrm{late.ar}}$):	$-18\text{€}/\mathrm{h}$
Marginal utility of leaving too early($\beta_{\mathrm{early.dp}}$):	$-6\text{€}/\mathrm{h}$
Marginal utility of staying too short($\beta_{\mathrm{short.dur}}$):	$-6\text{€}/\mathrm{h}$

These parameters were selected in order to match the typical Vickrey scenario values of $-6\text{€}/\mathrm{h}$, $-12\text{€}/\mathrm{h}$, and $-18\text{€}/\mathrm{h}$. Upon further reflection, one does, however, recognize that the above values in conjunction with our approach do not do this, since when traveling or waiting, one incurs the *additional* penalty of time that one could spend doing an activity, at the "cost" of $-20\text{€}/\mathrm{h}$ (opportunity cost). Section 2.7 discusses better values.

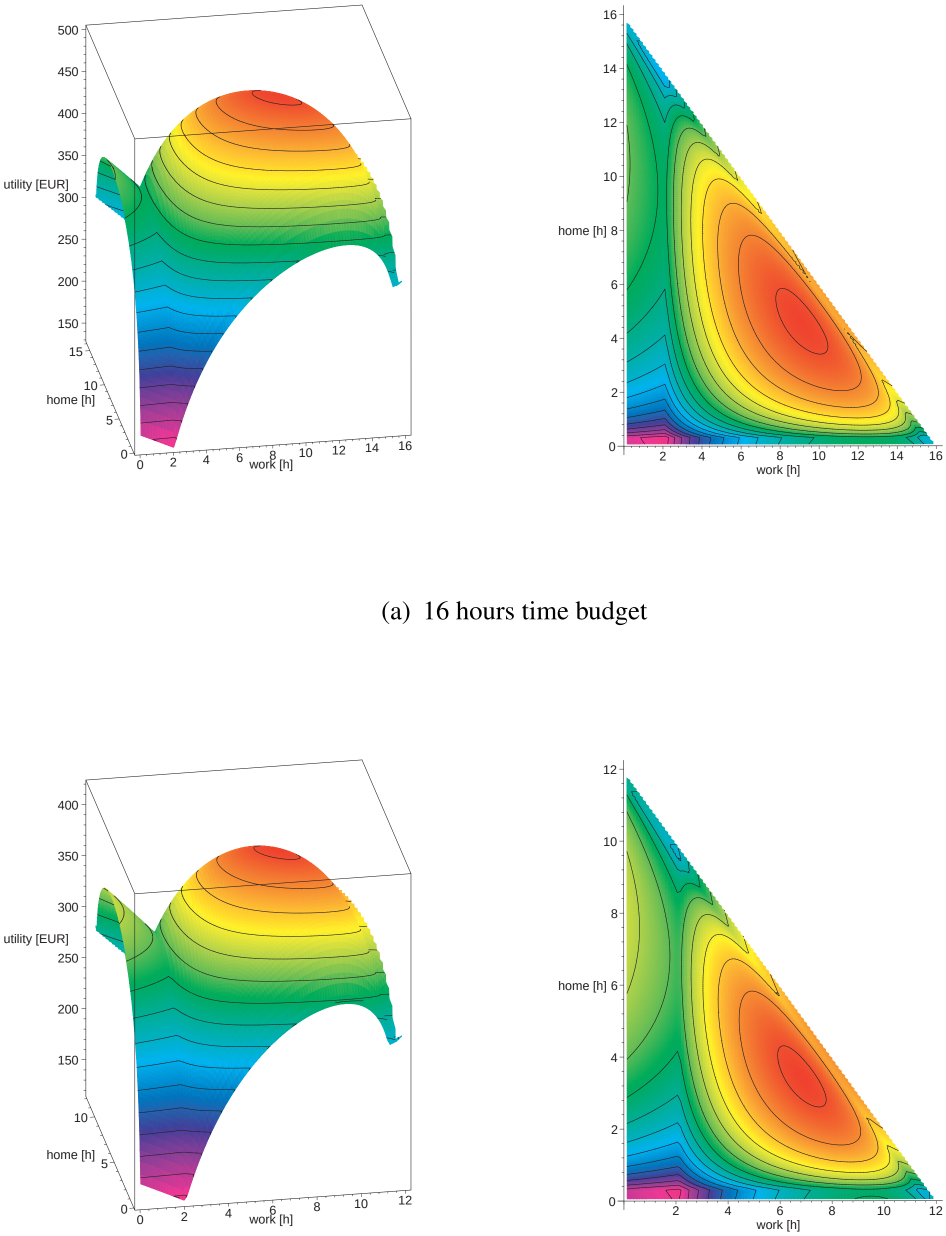
2.6.5 Examples of Utility Landscapes

In order to show some properties of the utility landscape, we exemplarily analyze plots of some activity patterns. Because of the limited number of displayable dimensions, our choice is restricted to very short activity patterns with a maximum of four activities.

Now, if we would use full length day plans with such a limited number of activities, the resulting day plans would be rather slack, and as a result of that, many of the typical problems with activity planning would not arise.

Our way out of this problem is to use shorter time budgets for our example activity plans. In consequence, the plans that we are going to talk about here are no longer complete day plans but rather partial plans. However, since the calculation of the utility values is completely additive, having a look at partial plans does make sense.

As a further simplification, we assume during this part of our investigations that all activities can be carried out at the same location. By applying this simplification, we do not have to define a lot of location related parameters. However, the problems that can be observed in this simplified context have the same complexity as with traveling enabled. This is due to the fact that our travel times are independent of time of day and because they are linear in the geometric distance of the locations.



(a) 16 hours time budget

(b) 12 hours time budget

Figure 2.3: Utility of activity pattern *home-work-leisure-home*. t^* of work, leisure and home set to 8, 2, and 4 hours, respectively.

We first show the typical utility landscape of an activity chain with four activities *home*, *work*, *leisure* and *home*. The t^* for *work*, *leisure* and *home* had been set to 8, 2, and 4 hours, respectively. There were no additional constraints that had to be fulfilled except that the total length of the plan was fixed to 16 hours (Figure 2.6.5).

The utility function as a function of the two parameters *duration of work* and *duration of leisure* has one clean global optimum. It should be easy to find the global optimum of this activity pattern even with standard optimization techniques. At the borders of the domain, there exist small regions where the utility increases again. This is due to the definition of the utility function which has a cutoff at very short durations. For example, when the duration of work becomes less than two hours, it becomes better not to work at all, in which case the additional leisure time makes a positive contribution.

In order to show the effect of the time budget on the time allocation, we show another plot of the same activity pattern but this time with a desired plan length of 12 hours (Figure 2.6.5). The optimum is shifted towards shorter times and the utility of the optimum is roughly 100 lower.

The utility landscape of activities with opening hours is more complex. As an example we show the utility of shopping in a store that is open during the morning from 9 until 11 and in the afternoon from 14 until 17. Without opening hours the landscape would be very smooth and completely independent of time of day. *With* the opening hours some interesting new properties can be observed.

In figure 2.4 we see that now there exist three local optima. The global optimum (top left) corresponds to showing up in the shop at 9 in the morning and staying there until 17 in the evening. The lunch break is spent waiting. The two local optima in the morning and the afternoon are much more meaningful. The early local optimum corresponds to coming at 9am when the shop opens and leaving the shop at 11am when it closes. The late local optimum corresponds to coming at 14 when the shop opens and leaving at 17 when it closes.

The structure of the fitness (utility) landscape becomes even more complex when we consider an activity *chain* that includes a complex activity with opening hours. Figure 2.5 shows the utility of the activity pattern *work-shop-work*. t^* was set to eight hours for the total working duration and two hours for the shopping activity. The overall plan was set

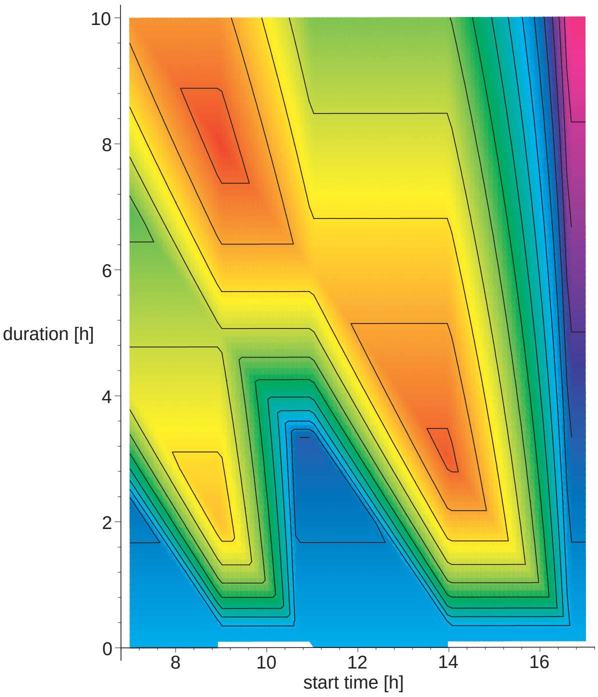
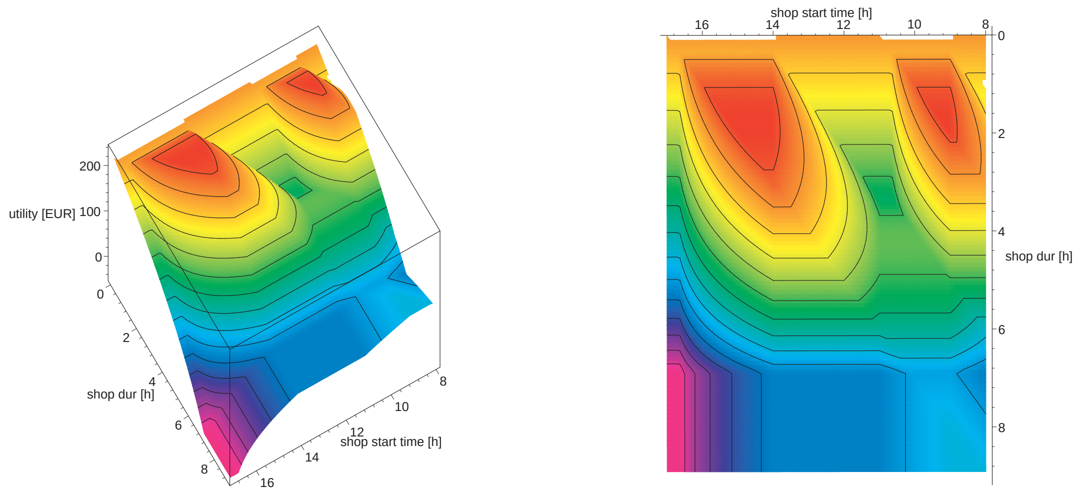


Figure 2.4: Utility of activity *shop*. Shop open 9–11, 14–17. There is no constraint on the total length of the partial plan.

to start at 8 and last until 17. One sees that good plans set the shopping time to 1.5 hours; the overall work time will then be set to 7.5 hours.

The utility for this activity pattern shows two equivalent global optima. They correspond to go shopping in the morning for almost two hours and to go shopping in the afternoon. The optimum in the afternoon is wider, because the shop is open for three hours in the afternoon.



(a)

(b)

Figure 2.5: Utility of activity pattern *work-shop-work*. The shop is open 9–11 and 14–17. For the calculation of the utility of work only the total working duration is regarded. That is, the sum of working duration before shopping and after shopping is first calculated and then the utility for work is calculated. Note also that the axes in this plot are reversed for visibility reasons.

2.7 Tests and Results for Complete Day-Plans

In this section, we want to show the general ability of our model to generate complete day plans. For that purpose, we designed a simple city. In our city there exist a number of different facilities including apartments, working places, shops, kindergartens and recreational areas. For each of these facilities, there exist three different locations between which an agent can choose. The different locations of the facilities are summarized in a map. See, e.g., Figure 2.7.

A possible oddity of our examples is that all locations can be changed on the same time scale. That is, even the home location can change between different solutions. It is, however, very easy to keep some of these things fixed by just reducing the choice in the corresponding category to one. This would also allow to run the algorithm at several levels, for example:

- Run the algorithm while long-term locations (e.g. home, work, kindergarten) are fixed in order to determine the short-term flexible parts of a daily schedule.
- Run the algorithm, say, for a possible new home location in order to test if an agent could improve its daily utility function by moving to this location (Abraham and Hunt, 2002). In this situation, the algorithm would model the "what-ifs" of humans when they consider alternative options without actually executing them.
- Clearly, any intermediate level is possible as well, for example when home and work are kept fixed but a kindergarten location is searched for.

For our tests we assume that travel times are proportional to the geometric distance between two locations and that the travel speed is constantly $10 \mathrm{km}/\mathrm{h}$.

We want to test our algorithm for different kinds of agents with different lists of activities that they would like to accomplish. For this purpose, we defined three different scenarios. Each scenario is defined by a set of activities that are available. The scenario "full10" is the one with the largest set and consist of ten activities which are listed in Table 2.1. The other two scenarios use a subset of these activities.

All facilities needed by the activities except for the facility "home" are not open during the whole day. The opening times used for our investigations are summarized in Table 2.2.

The three scenarios are defined as follows:

- As mentioned above, the "full10" scenario defines a scenario with set of ten activities. This scenario consists of the activities "sleep", "breakfast", "lunch", "dinner", "early work", "late work", "bring children to kindergarten", "fetch children from kindergarten", "shop", and "leisure". The purpose of this scenario is to show the performance of our algorithm if there are very many activities to perform.

Name	Priority	t^{*1}	$t_{\mathrm{latest.ar}}{}^2$	$t_{\mathrm{earliest.dp}}{}^3$	$t_{\mathrm{short.dur}}{}^3$	Facility
sleep	1	8.0	25.0	29.0	6.0	home
early work	1	4.0	9.0	11.0	3.5	work
late work	1	4.0	14.0	15.0	3.5	work
breakfast	3	0.5	10.0		0.25	home
lunch	2	1.25	14.0	12.0	0.75	work/home[4]
dinner	2	2.0	21.0	18.0	0.75	home
bring to kindergarten	1	0.25	9.0	8.5	0.25	kindergarten
fetch from kindergarten	1	0.25	16.0	15.5	0.25	kindergarten
shopping	3	2.0			0.5	shop
leisure	3	2.0			1.0	leisure

[1] t_0 (not shown in table) is derived from the priority and t^*. See Section 2.6.1 and Eq. (2.6).
[2] Latest starting time. If the activity starts later a penalty is applied. See Section 2.6.2
[3] Both $t_{\mathrm{earliest.dp}}$ and $t_{\mathrm{short.dur}}$ address the problem of not executing an activity enough long. If the activity is ended before $t_{\mathrm{earliest.dp}}$ a penalty applies. The same penalty applies if the duration of the activity is shorter than $t_{\mathrm{short.dur}}$. See Section 2.6.2
[4] Depending on the scenario, the facility of the activity "lunch" was either work or home. See description of the scenarios.

Table 2.1: Activities of our test case

Facility Name	Opening Times
Home	00:00-24:00
Work	06:00-20:00
Kindergarten	8:30-9:00 15:30-16:00
Shop	09:00-19:00
Leisure	14:00-24:00

Table 2.2: Opening times of the facilities: An activity can only be carried out when the required facility is open. Note facility "Kindergarten" has two opening times. The first one defines when children can be dropped off and the second when they can be picked up.

- In the "houseman" scenario, the agent has only eight of the ten activities of the "full10" scenario to choose from, leaving out both work activities. That is, the remaining activities are: "sleep", "breakfast", "lunch", "dinner", "bring children to kindergarten", "fetch children from kindergarten", "shop" and "leisure". Since in this scenario the agent does not have a work location, we changed the facility for eating lunch from “work” to “home”. This scenario simulates a rather dense day plan of a non working person.

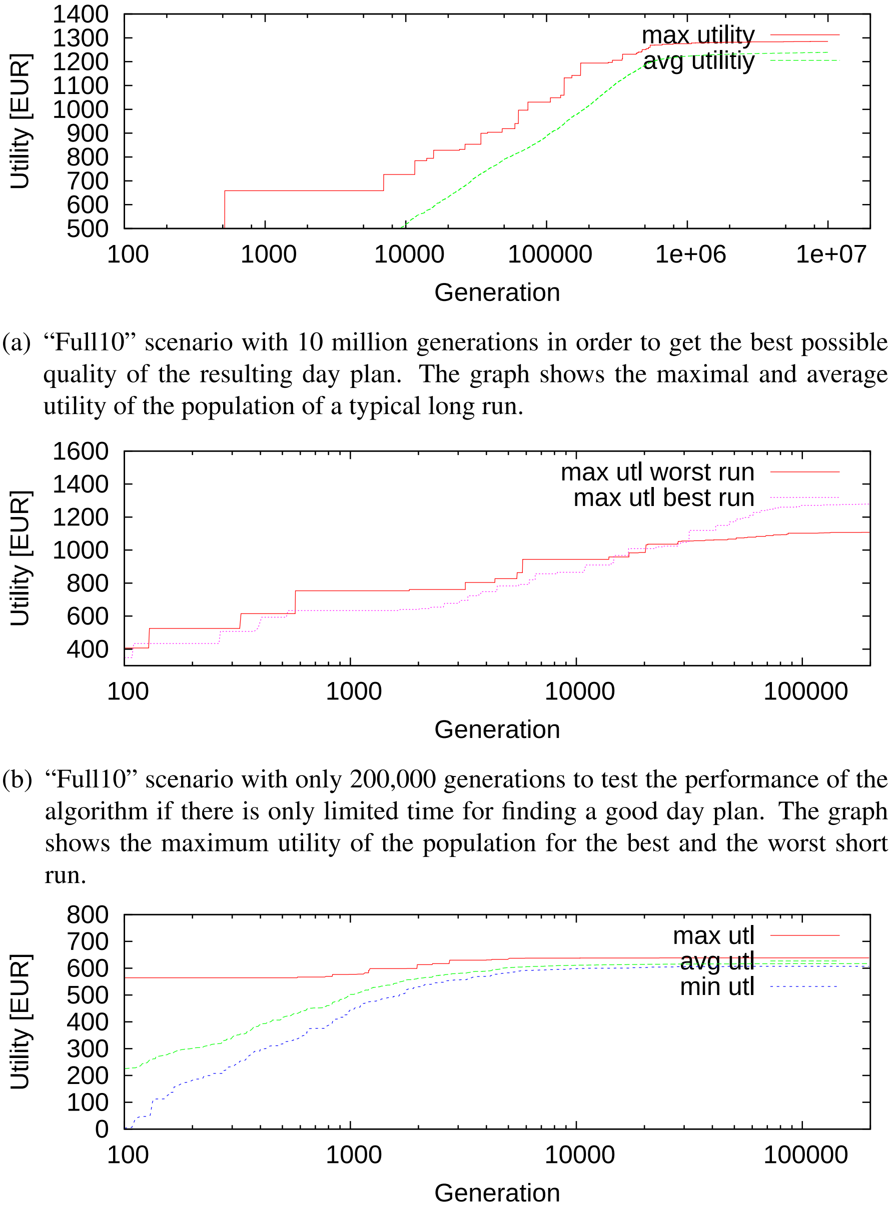
- In the “pensioner” scenario, the set of activities consists only of 5 activities; these are “sleep”, “lunch”, “dinner”, “shopping” and “leisure”. As in the “houseman” scenario, the facility for having lunch was changed to “home”. With this scenario we want to show how our model deals with day plans with a large freedom in time allocation.

In all these scenarios, the set of activities is endogenous to the model. This is discussed in Section 2.8.

We do two separate investigations for each scenario. In the first investigation, we run the algorithm for a long time in order to rate the quality of the best solution that the algorithm possibly produces. In the second investigation, we want to rate if the algorithm is capable of finding usable day plans within a limited time, therefore, we run the program for a short time. This second question is especially important if we want to generate day plans for a large number of agents.

For the quality investigation, we run the algorithm for 10,000,000 generations with a GA-population size of 300. The execution takes between 140 and 230 seconds on a 2.4GHz Pentium 4 Laptop. For the usability and speed investigation, we change the parameters in order to achieve a higher convergence speed taking into account that we sacrifice some stability for that reason. The GA-population size is reduced to 50 and the number of generations is limited to 200,000. Here, the execution takes between 3 and 5 seconds.

The quality test for the “full10” scenario was run with 5 different initializations. At the end of each run, the day plan with the highest utility value was always identical in terms of generated pattern and location selection. Only the time allocations differed, but the differences were within very few minutes. In Figure 2.6(a) we show a typical convergence graph for the quality tests. The best day plan found for the “full10” scenario in the quality test is shown in figure 2.7. Note that the children are not dropped off at home before continuing with the day plan; especially “leisure” is performed together with the children. This can be observed in almost all generated day plans. There is nothing that would force the



(a) “Full10” scenario with 10 million generations in order to get the best possible quality of the resulting day plan. The graph shows the maximal and average utility of the population of a typical long run.

(b) “Full10” scenario with only 200,000 generations to test the performance of the algorithm if there is only limited time for finding a good day plan. The graph shows the maximum utility of the population for the best and the worst short run.

(c) “Pensioner” scenario for a typical short run. In this test, the goal was to find a good day plan within limited time.

Figure 2.6: Convergence graphs for different learning scenarios

agents to drop children off before starting with the next activity. However, if desired, this could be easily included in the utility function.

In the five short runs for investigation of speed and usability for the scenario “full10” three times a solution was found that is identical to

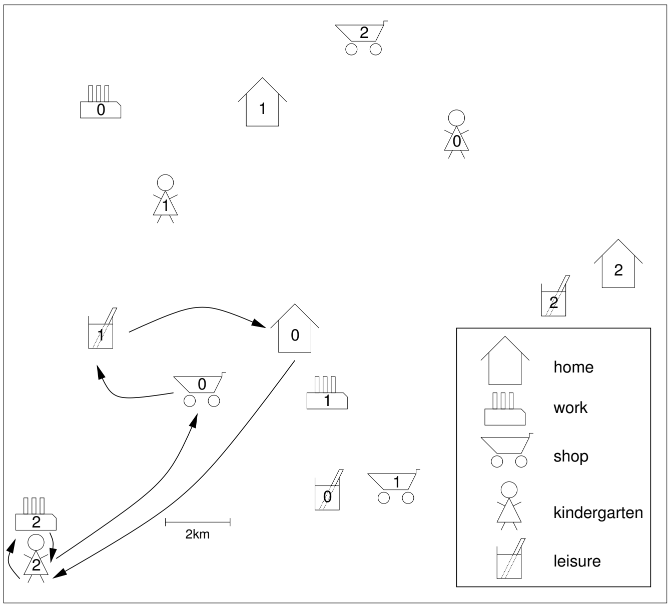


Figure 2.7: Map of the best day plan for the “full10” scenario after 10 million generations. The total utility is 1284.93. See Table 2.3 for the corresponding schedule.

the one shown in Figure 2.7 in terms of generated pattern and selected locations. Only the time allocations were not as sophisticated. The total utility varied between 1277.54 and 1278.84. In the two remaining runs, two different solutions were found. The solution shown in Figure 2.8 differs only in terms of location selection and time allocation while the solution shown in Figure 2.9 differs also in the generated pattern. Note that in this day plan the activity “bring children to kindergarten” is left out. In Figure 2.6(b) we compare the convergence of the best and the worst short run.

In all runs for the “pensioner” scenario (five long runs and five short runs) the same day plan was found with respect to the generated activity pattern and the selected locations. The time allocations to the different activities was also very similar. The similarity of the day plans can also be seen when looking at the total utility which was always between 638.483 and 638.514 —a very narrow interval. The only difference between the plans was their starting time which varied in the range from 10:33am to 11:45am. This is due to the lack of constraints for the activ-

Activity Name	Travel Time	Execution Time	Location
Breakfast		06:56–07:26	home 0
Bring children[1]	07:26–08:30	08:30–08:40	kiga[3] 2
Early work	08:40–08:46	08:46–11:52	work 2
Lunch		11:52–12:40	work 2
Late work		12:40–15:40	work 2
Fetch children[2]	15:40–15:46	15:46–16:00	kiga[3] 2
Shop	16:01–16:43	16:43–18:26	shop 0
Leisure	18:26–18:48	18:48–20:27	leisure 1
Dinner	20:27–21:03	21:03–23:02	home 0
Sleep		23:02–06:56	home 0

[1] Bring children to kindergarten.
[2] Fetch children from kindergarten.
[3] Kindergarten

Table 2.3: Schedule of the best day plan for the “full10” scenario after 10 million generations. The total utility is 1284.93. See Figure 2.7 for a graphical summary.

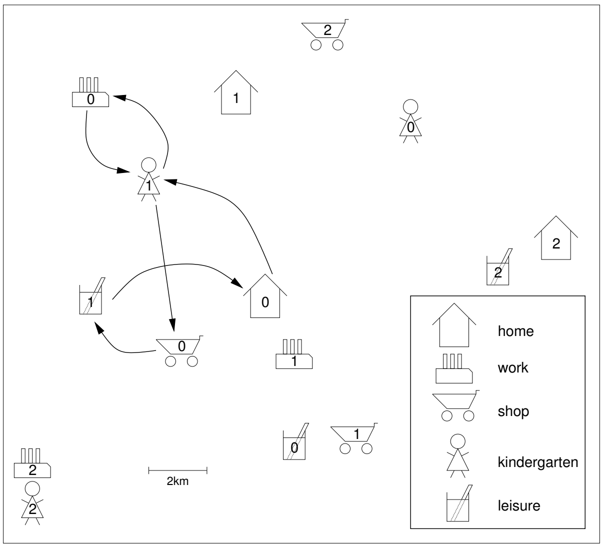


Figure 2.8: Map of the first alternative day plan for the “full10” scenario after 200,000 generations. The total utility is 1276.46. See Table 2.4 for the corresponding schedule.

Activity Name	Travel Time	Execution Time	Location
Breakfast		07:27–07:56	home 0
Bring children	07:56–08:30	08:30–08:42	**kiga 1**
Early work	08:42–09:03	09:03–11:54	**work 0**
Lunch		11:54–12:49	**work 0**
Late work		11:54–12:49	**work 0**
Fetch children	15:18–15:39	15:39–15:57	**kiga 1**
Shop	15:57–16:33	16:33–18:13	shop 0
Leisure	18:13–18:35	18:35–20:26	leisure 1
Dinner	20:26–21:02	21:02–23:30	home 0
Sleep		23:30–07:27	home 0

Table 2.4: Schedule of the first alternative day plan for the "full10" scenario after 200,000 generations. The total utility is 1276.46. See Figure 2.8 for a graphical summary.

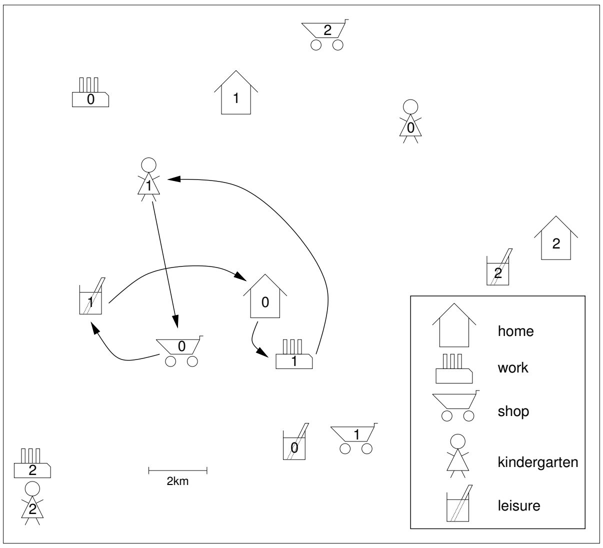


Figure 2.9: Second alternative day plan for the "full10" scenario after 200,000 generations. The total utility is 1107.89. Note, the activity "bring children to kindergarten" is left out. See Table 2.5 for the corresponding schedule.

Activity Name	Travel Time	Execution Time	Location
Breakfast		06:14–06:46	home 0
Early work	06:46–06:59	06:59–10:56	work 1
Lunch		10:56–11:59	work 1
Late work		11:59–15:04	work 1
Fetch children	15:04–15:51	15:51–16:00	kiga 1
Shop	16:00–16:37	16:37–18:26	shop 0
Leisure	18:26–18:47	18:47–20:27	leisure 1
Dinner	20:27–21:03	21:03–23:06	home 0
Sleep		23:06–06:14	home 0

Table 2.5: Second alternative day plan for the “full10” scenario after 200,000 generations: The total utility is 1107.89. Note, the activity “bring children to kindergarten” is left out. See Figure 2.9 for a graphical summary.

ities. That is, it does not matter at which time inside the range the day plan starts.

It seems that our algorithm has no problems to find a good solution for the “pensioner” scenario. In order to test this assumption we show the convergence graph of one of the usability runs in Figure 2.6(c). One can see that the algorithm converges already very early. In fact, a solution identical to the one found in the end in terms of generated pattern and selected locations is already found after 30,000 generations—which is only 15% of total number of generations.

In all five long runs for the “houseman” scenario and in four of the five short runs the same day plan was found with respect to the generated activity pattern and the selected locations. The time allocations were also very similar, they all lay within ten minutes. The resulting utilities vary between 1040.51 and 1043.04. We show the best day plan found in Figure 2.10.

The day plan found in the remaining short run is topologically different from the other day plans. It leaves out the activity “leisure” (not shown).

The resulting day plans for all three scenarios are plausible. All aspects of activity planning—activity pattern generation, location selection

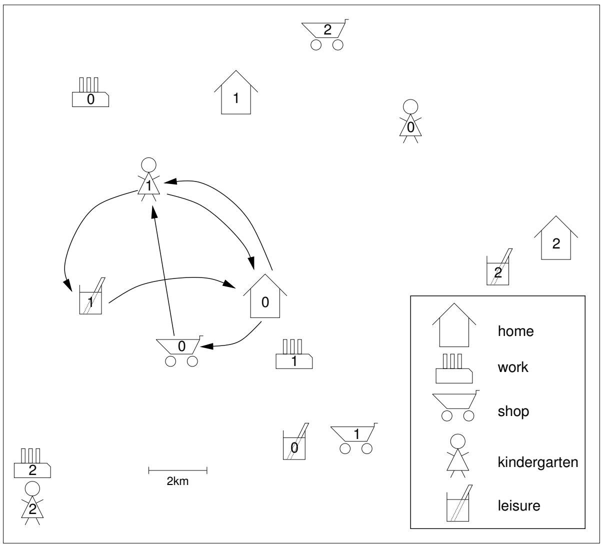


Figure 2.10: Map of the best day plan found for the "houseman" scenario. The total utility is 1043.04. Note, that the agent fetches the children and performs activity "leisure" with them. The reason why it does not drop them off at home before is that there is no constraint forcing it to do so and the presented day plan minimizes travel time. See Table 2.6 for the corresponding schedule.

Activity Name	Travel Time	Execution Time	Location
Bring children	08:08–08:42	08:42–08:59	kiga 1
Breakfast	08:59–09:33	09:33–10:17	home 0
Lunch		10:17–12:02	home 0
Shop	12:02–12:23	12:23–14:58	shop 0
Fetch children	14:58–15:35	15:35–15:52	kiga 1
Leisure	15:52–16:19	16:19–18:51	leisure 1
Dinner	18:51–19:27	19:27–22:02	home 0
Sleep		22:02–08:08	home 0

Table 2.6: Schedule of the best day plan found for the "houseman" scenario. The total utility is 1043.04. Note, that the agent fetches the children and performs activity "leisure" with them. The reason why it does not drop them off at home before is that there is no constraint forcing it to do so and the presented day plan minimizes travel time. See Figure 2.10 for a graphical summary.

and time allocation—are taken care of. The convergence of the genetic algorithm seems to be very stable, as for each scenario at the end of almost all test runs the same good solution is found. It seems, that our tests have not yet pushed our algorithm to its limits, it would be interesting to see how it performs on generating complete week plans.

2.8 Discussion and Further work

The values for the marginal utilities (Section 2.6.4) were selected to match typical values of the Vickrey scenario (e.g. Arnott *et al.*, 1993). It turns out, however, that in our framework the *effective* penalties of traveling or waiting are not the values of Section 2.6.4, but they are those values *plus* the utility lost by not doing any activity, i.e. $-20/\mathrm{h}$ (opportunity cost). This means that the *effective* marginal disutilities of traveling and waiting are $-32/\mathrm{h}$ and $-26/\mathrm{h}$, respectively. The correct values of our system to obtain the typical Vickrey penalties of -6, -12 and -18 for waiting, traveling, and being late, respectively, are thus

Marginal utility of any activity(β_{dur}):	$6€/\mathrm{h}$
Marginal utility of travel time(β_{trav}):	$-6€/\mathrm{h}$
Marginal utility of waiting(β_{wait}):	$0€/\mathrm{h}$
Marginal utility of coming late($\beta_{\mathrm{late.ar}}$):	$-18€/\mathrm{h}$
Marginal utility of leaving too early($\beta_{\mathrm{early.dp}}$):	$-18€/\mathrm{h}$

These are the values we will use in some of our future research.

An interesting consequence of the above observations is that it is not necessary to assume any disutility (negative utility) of travel at all. As long as there are activities whose marginal utilities are more strongly positive than the marginal utility of travel, agents will still attempt to minimize travel time. And if one wants to introduce a more sophisticated utility function for travel, for example first positive and then becoming negative, then this could be easily done. This means, in particular, that our approach will not have any problems with the recently discussed “positive utility of travel”. Clearly, such work should be done in conjunction with survey data that can then be used to estimate and test models.

In all scenarios of Section 2.7, the set of activities is endogenous to the model. Such sets could for example come from separate models of activity repertoires. However, our algorithm is in principle capable to drop low priority activities, although that was not systematically investigated. Therefore, one could imagine to start from a maximum set of activities, possibly with the desire of intermediate home stops between any two activities, and then have the algorithm drop intermediate home stops and/or low priority activities. The priorities could even be coupled to household or weekly agendas, discussed below.

Genetic algorithms are known as rather inefficient, but very flexible search methods. This flexibility means that extensions can be easily introduced. Examples for such possible extensions are:

- The present model for the estimation of travel times between different locations is not very realistic. It would be possible to use travel times from a simulation in order to increase realism. With time dependent travel times, phenomena like congestion avoidance within the day plans should emerge.

- The paper pretends that all activity scheduling problems can be solved with a single utility function. This is improbable, and variations of the relevant coefficients could be easily introduced. Such coefficients could for example be obtained from estimated econometric models of daily activity schedules (e.g. Jara-Diaz and Guerra, 2003). That paper also explains how such econometric explanations relate to random utility theory.

- Aspects of limited information could be modeled by only making subsets of possible locations available to the algorithm. Such sets would need to be generated by other methods, for example by random sampling, by information transmission via social networks (Marchal and Nagel, 2005), or by using mental maps (Arentze and Timmermans, 2005; Kistler, 2004).

- The paper assumes that each agent optimizes for him-/herself. This reflects our observation that simple copying of partial strategies from other people does not work in the real world. For example, I cannot simply copy the shopping location from my neighbor because in all probability he/she works somewhere else and so this may not be convenient. A possible approach might once more be the use of social networks (Marchal and Nagel, 2005) which ensures that information is primarily passed between people who share some similarities.

- One interesting aspect of activity planning is to generate activity plans for households and/or for full weeks instead of generating them for single agents and single days. This problem is more complex than simply generating individual day plans for each of the occupants of a household since the activity plans of different occupants or of consecutive days depend on each other. For instance, only one of the agents living in a household has to go shopping, or shopping only needs to be done once a week. An investigation of this has been done by Meister *et al.* (2005b).

  More complicated would be the interaction of agents that do *not* live in easily identifiable groups, such as the coordination of shared rides or of leisure activities. Here, some other method would first have to provide the social links (e.g. Marchal and Nagel, 2005). However, even once social links are known, the GA approach would probably fail since too many agents would have to be coordinated at the same time. A sequential activity planning process, such as described by Doherty and Axhausen (1998), possibly combined with a GA, might be a possible approach.

2.9 Summary

A genetic algorithm (GA) was presented that constructs all-day activity plans. It uses as input a set of possible activities, and a utility function to score activity schedules. The GA attempts to construct good solutions which maximize the utility function. It does that by maintaining a *population* of solution instances, which are mutated, and which, importantly, breed new solutions by crossover. Crossover means that two solution instances are selected, and part of the new solution comes from one parent, and the other part from the other parent.

The algorithm is then run on several examples. It is shown that the algorithm generates plausible solutions both for crowded and for relaxed activity sets, and that it can do so even when the computation time is restricted.

The most important aspect of this work is that arbitrary utility functions can be used. A GA does not guarantee optimal solutions, but it will nearly always generate plausible solutions. Given that humans do no better, this may be sufficient for many travel forecasting purposes.

Acknowledgments

Kay Axhausen helped us with numerous pointers to all-day utility functions. Nevertheless, the responsibility for our choices remains with us.

Chapter 3

Q-learning for Flexible Learning of Daily Activity Plans

Authors

David Charypar
Institute for Transport Planning and Systems, ETH Zurich, Switzerland
Email: charypar@ivt.baug.ethz.ch

Kai Nagel
Institute for Land and Sea Transport Systems, TU Berlin, Germany
Email: nagel@vsp.tu-berlin.de

Publication

This paper was published in a journal, see Charypar and Nagel (2006).

Abstract

Q-learning is a method from artificial intelligence to solve the reinforcement learning problem (RLP), defined as follows: An agent is faced with a set of states, S. For each state, there is a set of actions, $A(s)$, which the agent can take and which takes the agent (deterministically or stochastically) to another state. For each state, the agent receives a (possibly stochastic) reward. The task is to select actions such that the reward is maximized. Activity generation is for demand generation in the context of transportation simulation. For each member of a synthetic population, a daily activity plan needs to be found, stating a sequence of activities (e.g. home - work - shop - home), including locations and times. Activities at different locations generate demand for transportation. Activity generation can be modeled as RLP with the states given by the triple (type of activity, starting time of activity, time already spent at activity). The possible actions are either to stay at a given activity, or to move to another activity. Rewards are given as *utility per time slice*, which corresponds to a coarse version of marginal utility. Q-learning has the property that, by repeating similar experiences over and over again, the agent looks forward in time, i.e. the agent can also go on paths through state space where high rewards are only given at the end. This paper presents computational results with such an algorithm for daily activity planning.

Keywords: Activity Planning; Q-Learning; Activity Generation; Within Day Replanning; Multi Agent Travel Simulation

3.1 Introduction

It is a recent trend in transportation research to use activities in order to generate demand for transportation. Transportation demand is naturally derived from performing activities at different locations. For each synthetic individual, a sequence of activities is generated, including activity location and activity times. Activity-based demand generation is a very active field of research. The mainstay of activity-based demand generation are random utility models (RUMs) (Ben-Akiva and Lerman, 1985; Bowman *et al.*, 1999; Pendyala, 2004; Bhat *et al.*, 2004). RUMs, however, arguably have the disadvantage that (a) they are behaviorally not very realistic, (b) only heuristic approaches exist for reducing the *choice set* when faced with a very large number of options, (c) they need to be re-computed when a traveler is thrown *off* its optimal dayplan. In consequence, alternatives are also investigated, such as behaviorally or rule-based approaches (e.g. Arentze *et al.*, 2000; Miller and Roorda, 2003).

This paper investigates if a certain approach from machine learning, called *Q-learning*, is applicable to activity generation. The two main questions to answer at this state are: (a) Are the results plausible? (b) Can they be obtained within reasonable time on a computer? A more expansive research only makes sense if both questions can be answered in the affirmative.

This paper concentrates on a module to generate the *time allocation* part of activity plans, i.e. when activities should begin, how long they should last, and when they should end. The method should, however, also be capable to do location choice, or activity pattern selection. Some discussion of this, and related methods that could be explored, can be found in Section 3.6.

This paper starts with a review of Q-learning (Section 3.2), followed by a section on how to apply this to activity generation (Section 3.3). The method is first tested on an unrealistic but somewhat challenging *test* example, and then with a somewhat more realistic case (Section 3.5). The paper is concluded by a discussion (Section 3.6) and a summary (Section 3.7).

3.2 Q-Learning

The reinforcement learning problem (RLP) can be stated as follows: Given a set S of states s, transitions between states $s \to s'$, and rewards $R(s \to s')$ associated with each transition. At each state, the agent can select between different actions $a \in A(s)$, which influence the transition probabilities between states. The task of the agent is to select actions $a(s)$ such that some expected discounted reward,

$$E\left(\sum_t \beta^t R_t\right) , \tag{3.1}$$

is maximized. $\beta < 1$ is the discount factor, and R_t is the reward being obtained at each time step t.

Rewards can be high, low or even negative. They may come with a delay, in the sense that some transition may not lead to immediate high reward, but possibly to a high later reward which cannot be reached by any other sequence of transitions.

β models the effect of how far the agent looks into the future. A β close to one means that rewards in the far future carry large weight; a small β means the opposite.

Q-learning (Russell and Norvig, 1995, e.g.) is a method to solve the RLP. In Q-learning, an agent learns action-values giving the expected utility of taking a certain action in a given state. These action-values are also called Q-values. In each state, the agent has several options for actions it could execute. For each state-action-pair, the agent stores an individual Q-value $Q(s, a)$ that is used for the decision process. The Q-value for a given state-action-pair corresponds to the expected cumulative reward, Equation 3.1, that can be collected by taking the action.

Q-values are defined the following way:

$$Q_\infty(s, a) = R(s') + \beta \max_{a'} Q_\infty(s', a') , \tag{3.2}$$

where $R(s')$ is the reward for arriving at s', and s' is the state that is the result of executing action a in state s. If the transition after taking s is probabilistic (i.e. several different s' can result), then some suitable average needs to be taken. β is again the discount parameter, in the range [0..1[.

Equation 3.2, when expanded, leads to a solution of type $\sum_t \beta^t R_t$. If an agent always takes the action $a \in A(s)$ which maximizes $Q_\infty(s,a)$, then the agent maximizes that sum. When deviating from that *path*, the reward will be reduced. This shows that always taking the action with the highest Q-value solves the RLP.

Note that for low discount parameters, Q_∞ depends almost entirely on the current state action pair, resulting in a very *greedy* search for the optimal solution. In contrast, large discount values correspond to very long time horizons which means that the agent can look ahead and make its decisions on more global reflections. Some further insight is gained by assuming, for the moment, constant rewards $R(s) \equiv \bar{R}(\forall s)$. In that situation, all Q_∞ need to be the same, and therefore $\tilde{Q}_\infty = \bar{R} + \beta\,\tilde{Q}_\infty$ and thus

$$\tilde{Q}_\infty = \frac{1}{1-\beta}\,\bar{R}\,. \tag{3.3}$$

By this argument, one sees that the final Q-values are proportional to the average reward, and the proportionality factor is $\frac{1}{1-\beta}$. For $\beta <\sim 1$, the factor is very large, and for $\beta \to 1$, it goes to infinity.

So far, we have only described the steady state of Q-learning, that is the final solution *after* learning. However, the steady state is not initially known, and therefore, it is crucial to have a look at the actual learning process. The algorithm that we use in order to approximate the steady state (the actual Q-learning algorithm) is the following:

1. Initialize the Q-values.

2. Select a random starting state s which has at least one possible action to select from.

3. Select one of the possible actions. This action will get you to the next state s'.

4. Update the Q-value of the state action pair (s,a) according to the update rule (Equation 3.4) below.

5. Let $s = s'$ and continue with step 3 if the new state has at least one possible action. If it has none go to step 2.

The update rule is given by

$$Q_{t+1}(s, a) = (1 - \alpha)\, Q_t(s, a) + \alpha\, [R(s') + \beta \max_{a'} Q_t(s', a')], \tag{3.4}$$

where $Q_t(s, a)$ is the Q-value at the current time-step and $Q_{t+1}(s, a)$ is the updated value. α is the learning rate and is a parameter of the algorithm. The proper selection of α is crucial in many applications. Watkins and Dayan showed that the Q-learning algorithm converges to the steady state if α converges to zero with certain mathematical properties (Watkins and Dayan, 1992). However, in our cases we achieved good results with $\alpha = 1$, which is probably because we have deterministic rewards.

In each state the agent basically can choose from two kinds of behavior: either it can explore the state space or it can exploit the information already present in the Q-values. By choosing to exploit, the agent usually gets to states that are close to the best solution so far. By this it can refine its knowledge about that solution and collect relatively high rewards. On the other hand, by choosing to explore the agent visits states that are farther apart from the currently best solution. By doing so, it is possible that it finds a new, better solution than the one already known.

We use a parameter - the exploration rate p_{explore} - to set the behavior of our Q-learning algorithm. In every step, with a probability of $1 - p_{\text{explore}}$ the agent exploits the information stored in the Q-values, with probability p_{explore} the agent chooses a random action in order to explore the state space.

Now assume that the agent has learned some Q-values for this situation, either Q_∞ according to Equation 3.2 or some other values. Assume that now exploration is switched off (i.e. $p_{\text{explore}} = 0$), that is, at every state s the agent selects the action a which maximizes $Q(s, a)$. Can one say anything about the long-term behavior in this situation?

In fact, this describes a discrete dynamical system. Let us also assume that the number of state action pairs is finite. Since this is now a *deterministic* system, the trajectory needs to go to an attractor, which is either a fixed point or a cycle. This is due to the following reason: as the system is discrete and finite, the trajectory eventually needs to come back to a state where it was before; since the system is deterministic, from then on it will do exactly the same as in the previous cycle.

3.3 Q-learning for Daily Activities of Humans

How can the problem of daily activity planning be encoded in a way that it becomes a RLP? For this, assume that the day is segmented into a number of time slices, $t = 1..T$; this paper will investigate time slices of 15, 30, and 60 minutes. Possible states at each time slice are possible activities, e.g. *Home*, *Work*, *Shop*, etc. At each time slice, the agent needs to decide if it stays at the current activity, or moves on to a different one. If the sequence of the activities is fixed, then this is a binary choice between *stay* and *switch*; otherwise, it gets a bit more complicated.

To make the model realistic, a state needs to consist of the activity itself, the current time-of-day, as well as the duration that the agent has already spent at that activity. Activity type and current time only are not sufficient: being at work at 3pm but having arrived at 8am is different from being at work at 3pm but having arrived at 11am. These rewards are easiest defined in terms of reward tables for each activity, which, for each arrival time and each duration, give the reward for staying one more time slice. If the reward is taken as utility, then the reward tables are the same as a discretized marginal utility, multiplied by the duration of a time slice.

The time structure is assumed to be periodic, that is, at $t = T$ the agent is connected to $t = 1$, and over the transition it can stay or switch as it can do with any time increment. Now assume once more that the agent has learned some Q-values and now does exploitation only, i.e. it always chooses the action a which maximizes $Q(a, s)$ at any state s. Because of the time structure, the system cannot go to a fixed point, and so under normal circumstances it will describe a cycle through the 24-hour state space. If the RLP was completely solved, then for $\beta \to 1$ that path maximizes the score (utility) per 24 hours. For smaller β, the situation is similar, but not exactly the same. If the RLP was not completely solved, i.e. some of the Q-values do not correspond to the steady state values, then the agent will nevertheless find a cycle, albeit possibly not the optimal one.

Note that those cycles can also be multiples of 24 hours. For example, an agent can have one full day where it gets up early and goes to bed late, alternated with a less full day where it gets up later and goes to bed earlier.

An interesting side-effect of the structure of Q-learning is that the result of the computation is not only the optimal *cycle* through state space, but also the optimal *paths* if the agent is pushed away from the optimal cycle. For example, if a transfer between activities takes considerably longer than expected, the Q-values at the arrival state will still point the way to the best continuation of the plan.

3.4 The *Test* Example

3.4.1 Description

For testing purposes, the following task was designed: given an activity pattern of four activities - *Home*, *Work*, *Shop* and *Leisure* - we want to solve the time allocation problem, i.e. find starting and ending times for these 4 activities such that the resulting overall utility is maximal; the overall utility is thereby defined as the sum of the utilities of the individual activities.

We define the utility function of the activity *Home* to be independent of the starting time. It is a step function of the duration with the step being at seven hours. The utility of being at home for less than 7 hours is defined as being zero, the utility of staying at home for seven hours or longer is 7.

The activities *Shop* and *Leisure* both use the same utility function. It is of the same type as the one of *Home* but the step is at 2 instead of 7 hours. Also the height of the step is set to 2 instead of 7.

The utility function of activity *Work* - other than the ones above - is starting time dependent. Only if the agent starts working at 8:00 in the morning and stays there for at least 9 hours it will get a utility of 9. If it starts earlier or later or if it stays for a shorter time the utility of *Work* will be 0.

We have intentionally designed the utility functions above in a way that it is difficult to find the optimal solution. In order to do so, the agent has to look very far in the future as the rewards for correct behavior are not given until the end of an activity. Integrating activity *Work* into the daily plan is even more difficult as not only the duration of the activity has to be long enough but also the starting time has to be chosen correctly. If the agent decides to start working only 15 minutes earlier - or later - it will lose all the reward given in case of work starting at 8:00.

These reward tables describe a *difficult* case for the search algorithm, since there is no indication at all that a certain path will lead to a good reward later. However, these reward tables do not model reality. More plausible reward tables for activities generally are smoother and give rewards already at earlier times of activity execution.

In order to keep the number of states finite, the maximum duration of any activity is restricted to 12 hours. Depending on the resolution this is equivalent to setting the maximal number of consecutive states spent in the same activity to 12, 24, or 48, respectively.

Finally, we have to specify how we derive the reward tables from the utility functions. This can be done by calculating the discretized version of the marginal utility function. This is equal to the differences between utility values at consecutive positions in the utility function of a particular activity. These consecutive positions are placed according to the time resolution chosen.

To account for the fact that an agent usually has to travel from one location to another between activities we define a constant travel time between different activities. We choose it to be 60 minutes. The utility of travel is set to be zero. One might argue that the utility of travel should be negative. However, as all our activities have positive utilities per time it is already desirable to minimize travel time s. As mentioned earlier, there are basically 3 parameters playing a role for Q-learning: the discount parameter β, the learning rate α and the exploration rate p_{explore}.

The first problem we would like to solve is to find a reasonable value for the discount parameter β. In principle we would like to choose a value close to one for this parameter as we are interested in finding the day plan which maximizes the cumulative reward or utility. For the utility of a plan it does not matter *when* a certain reward is earned only *that* it is earned. As a result the discount parameter that corresponds best to the problem is $\beta = 1$. Unfortunately, this leads to diverging Q-values.

On the other hand, for efficiency, low discount parameters are best as they reduce interdependency of the Q-values and therefore lead to higher learning speeds. But, low discount parameters inherently prefer short activities. This can be to an extent that long activities (such as *Work*) are left out completely while short activities (such as *Shop*) are repeated over and over again.

Knowing all that, one has to find a compromise. Our preliminary tests showed that for time-slots of 60 minutes a discount parameter of $\beta = 0.96$ works fine. When the time resolution increases, the length of the activities in terms of number of states also increases and, as a consequence, the discount parameter β has to be increased accordingly. In order to have comparable results at different time resolutions, we want the discount per time t to be the same for all times $t > 0$ independent of the time resolution t_{res}. This leads to

$$\beta_{\text{res}_1}^{(\frac{1}{t_{\text{res}_1}})} = \beta_{\text{res}_2}^{(\frac{1}{t_{\text{res}_2}})}, \tag{3.5}$$

and then

$$\beta_{\text{res}_2} = \beta_{\text{res}_1}^{(\frac{t_{\text{res}_2}}{t_{\text{res}_1}})} . \tag{3.6}$$

For β_{res_1} close to 1.0 and $t_{\text{res}_2} < t_{\text{res}_1}$, one can approximate this by

$$\approx 1 - (1 - \beta_{\text{res}_1})\frac{t_{\text{res}_2}}{t_{\text{res}_1}}. \tag{3.7}$$

With $\beta_{\text{res}_1} = 0.96$ and $t_{\text{res}_1} = 60\text{min}$ one obtains

$$\beta_{\text{res}_2} = 1 - \frac{0.04 t_{\text{res}_2}}{60\text{min}}, \tag{3.8}$$

where t_{res_2} is the desired time resolution in minutes.

Since our reward tables are completely deterministic, we choose the learning rate α to be 1.0. This leads to the highest possible convergence speed. However, depending on the value of the discount parameter, the convergence can be slow.

The initialization of the Q-values has a large effect on the learning speed and the quality of the result. In general, initializations with high initial Q-values lead to more exploration of the state space as the agent has to find out first for each state that it has actually a lower Q-value. Accordingly, low initializations lead to less exploration. If the Q-values are initialized to low values, the agent finds one feasible solution very soon and sticks with it very long. As some kind of a compromise, random initialization in a reasonable range is very often used.

High and *low* are defined with respect to the final Q-values $Q_\infty(s, a)$: a high initial Q-value is larger than any final Q_∞, a low initial Q-value is smaller than any final Q_∞. Some a priori estimates can be obtained from Equation 3.3: Q-values are certainly high when they are above $\frac{1}{(1-\beta)R_{\text{max}}}$, where R_{max} is the largest reward in the system. Q-values are certainly low when they are below $\frac{1}{(1-\beta)R_{\text{min}}}$, where R_{min} is the smallest reward in the system. This also makes clear that *high* and *low* depend on the particular value of $beta$ that was selected.

For our problem we used a *high* initialization with Q-values of 30. This value is chosen such that it is higher than the highest Q-value in the steady state resulting in a complete exploration of the state space.

The last parameter that we have to deal with is the exploration rate p_{explore}. As exploration is basically already taken care of by our initialization, we decided to use a rather low exploration probability of $p_{\text{explore}} = 0.01$.

3.4.2 Results

The above scenario was tested with different resolutions. In the coarsest test we used a time resolution of 60 minutes with a discount parameter β of 0.96. In the subsequent tests the resolution was increased by factors of two resulting in 30, and 15 minutes respectively, with corresponding β values of 0.98 and 0.99 according to Equation 3.8.

From the design of the utility functions for the four activities *Work*, *Shop*, *Leisure* and *Home*, it follows that the optimal daily plan corresponds to the one shown in Table 3.1. Note that the time between activities is needed for traveling from one location to another.

We now compare the solutions found by the algorithm with the optimal solution shown in table 3.1. On top of that, we look at the learned Q-values and appraise the ability of the solution to recover from distur-

work	shop	leisure	Home
08:00-17:00	18:00-20:00	21:00-23:00	00:00-07:00

Table 3.1: Optimal day plan for the *test* example

Resolution t_{res}	Number of Iterations	Running Time
60 minutes	50k	100ms
30 minutes	500k	546ms
15 minutes	5M	4.89s

Table 3.2: Computational performance with the *test* example using *high* initialization: We show the number of iterations and the running time necessary to converge to the optimal solution, shown in Table 3.1, with a probability substantially higher than 50%.

bances. Only if fast and plausible ways of recovering are observed we assume that the algorithm has converged to a good solution.

In table 3.2 we show computational results obtained by our tests. It can be seen that doubling the resolution leads to an increase in the number of iterations needed to converge by a factor of 10. The table gives the minimal number of iterations needed in order to converge with a probability higher than 50%. We considered the algorithm as having converged if both the optimal daily plan corresponded to the one shown in table 3.1 and the agent was able to recover from disturbances in a reasonable manner. The running times were measured on a Mobile Pentium 4 with 2.4 GHz. Our programs were implemented using Java.

One might worry about reliability of the algorithm. As was already mentioned, table 3.2 says only something about the number of iterations needed in order to converge with a probability of at *least* 50%. What if we need a system that converges in at least 99% of the cases? It might be necessary to increase the number of iterations to 10 times the indicated value or even more. Fortunately, it does not seem to be this way. In fact, we never observed a failure to converge if the algorithm was run for twice the number of iterations stated in the table.

3.5 A *More Realistic* Example

3.5.1 Description

The data used for our *test* example is not very realistic. The *test* example was explicitly designed to be difficult to solve: since a reward for a particular activity is always given only at the end of the activity, the agent has to look far into the future.

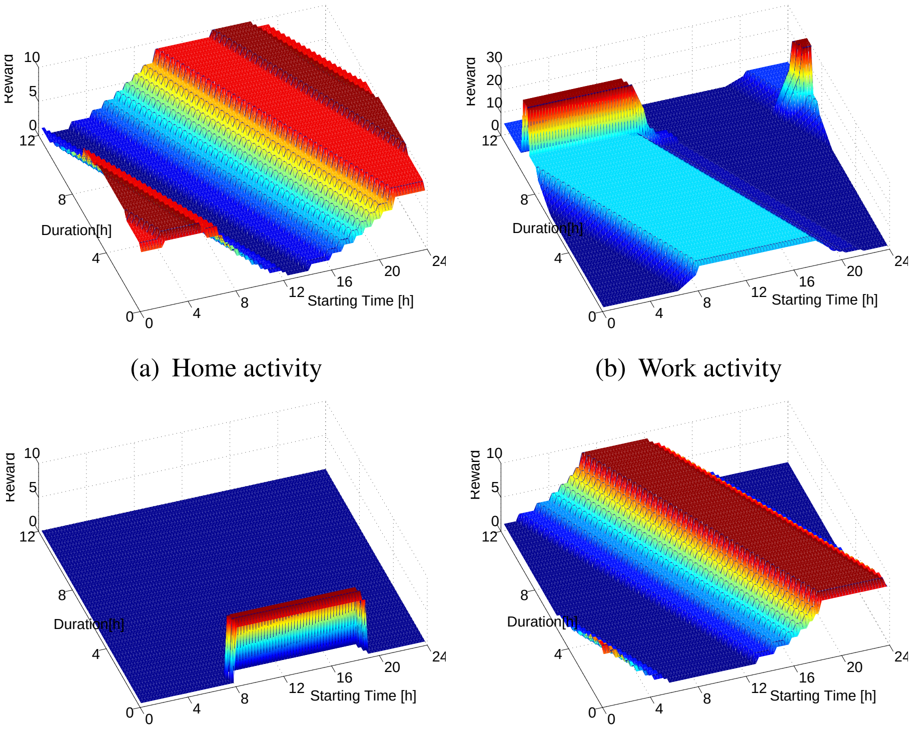
However, real life is different. Real activities already give rewards at earlier times. For example, being at home pays off already very early which corresponds to a reward table for the activity *Home* that has some positive rewards for each hour that it is executed. Therefore, we introduce new reward tables for all of the four activities:

- Activity *Home* (Fig. 3.1(a)): The reward for spending one hour at home is independent of the time that was already spent there. The reward only depends on the time of day, assuming that being at home during the night (sleeping) pays off more than being at home during the day.

- Activity *Work* (Fig. 3.1(b)): Work pays off most from 8:00 until 18:00. If the agent performs work outside this time window the rewards per time slice get gradually reduced. The agent gets a bonus for staying at work for more than 9 hours. However, the reward is reduced if 10 or more hours are spent at work. Working between 21:00 and 7:00 does not give any reward at all.

- Activity *Shop* (Fig. 3.1(c)): We assume that shops are open from 8:00 until 19:00. Therefore, shopping gives only rewards during the day. Maximal shopping time is set to 45 minutes. If an agent shops for a longer time, it does not get any reward for the additional time.

- Activity *Leisure* (Fig. 3.1(d)): We define leisure similar to activity *Home*. The reward for having one time slice of leisure is maximal from 19:00 until 24:00 and independent of the time already spent in this activity. It is minimal, although not zero, from 5:30 until 13:00. There is a smooth transition in between.



(a) Home activity

(b) Work activity

(c) Shop activity

(d) Leisure activity

Figure 3.1: Plot of the rewards per 15 minutes for different activities in the *more realistic* example: Rewards depend on the starting time of an activity and the duration that it lasted so far.

			t_{travel} for...	
From	To	...$t_{\text{res}} = 15\text{min}$	...$t_{\text{res}} = 30\text{min}$	...$t_{\text{res}} = 60\text{min}$
Home	Work	45min	60min	60min
Work	Shop	15min	30min	60min
Shop	Leisure	15min	30min	60min
Leisure	Home	30min	30min	60min

Table 3.3: Travel times for the *more realistic* example, assumed to be constant throughout the day.

Also the travel times between activities were changed, see table 3.3. This was done in order to become comparable with earlier work of the Group for Simulation and Modelling (Graf, 2003).

Resolution	work	shop	leisure	home
60min	08:00-17:00	18:00-18:00 (skipped)	19:00-23:00	00:00-07:00
30min	08:00-17:30	18:00-18:30	19:00-23:30	00:00-07:00
15min	08:00-17:30	17:45-18:15	18:30-23:45	00:15-07:15

Table 3.4: Optimal day plan for the *more realistic* example

As with the *test* example, the algorithm was tested with the new reward tables and travel times with time resolutions of 60, 30 and 15 minutes. All Q-learning parameters were set to the same values as in the *test* example.

In search of a planning algorithm which is as fast as possible we also tried to use a low initialization approach. As mentioned earlier, Q-learning with low initialization quickly finds feasible solutions at the expense of slower convergence to the optimal solution. So if one is looking for reasonably good results and does not depend on optimal solutions this might be preferable.

With low initialization, we have to make sure that exploration is taken care of by other means. We therefore use higher exploration rates in this case as is also indicated in table 3.6. Depending on the time resolution we use an exploration rate p_{explore} of 0.4, 0.2 or 0.1 respectively.

3.5.2 Results

First we ran the Q-learning algorithm multiple times for a long time in order to reliably find a good daily plan for the given reward tables. This was done for each resolution independently. The resulting solutions are shown in table 3.4. Further on, we will refer to these solutions as the optimal solutions.

Similar to the *test* example, we test the algorithm for time resolutions of 60, 30 and 15 minutes respectively and identify the number of iterations needed in order to converge to the best solution with a probability greater than 50%. This we do both for the high initialization approach and the low initialization approach. See table 3.5 for results using the

Resolution t_{res}	Number of Iterations	Running Time
60 minutes	50k	143ms
30 minutes	500k	545ms
15 minutes	2M	1.98s

Table 3.5: Computational performance with the *more realistic* example using *high* initialization: We indicate the number of iterations needed to converge to the optimal solution with a probability substantially higher than 50%.

Resolution t_{res}	p_{explore}	Number of Iterations	Running Time
60 minutes	0.4	10k	78ms
30 minutes	0.2	50k	114ms
15 minutes	0.1	500k	559ms

Table 3.6: Computational performance with the *more realistic* example using *low* initialization: We indicate the number of iterations needed to converge to a *reasonable* solution with a probability substantially higher than 50%. p_{explore} is the exploration rate.

high initialization and table 3.6 for results of the low initialization approach.

The number of iterations necessary to converge for the real world example with high initialization and the *test* example are almost the same. This makes sense as the high initialization results in a complete exploration of the state space. As the state space is of the same size in both cases, it is to be expected that the number of iterations needed to explore it is roughly the same. Only at the highest time resolution there is an observable difference: the real world example converges to the best solution already after 2 million iterations compared to 5 million iterations for the *test* example.

Compared to the high initialization tests, with low initialization daily activity plans can be generated much earlier. It seems that usable plans together with reasonable disturbance recovery are produced already after approximately 10% of the time needed using the high initialization approach. However, these plans are not exactly the best daily plans identified by using the explorative initialization. Here, the engineer has the freedom to choose the method which better suits his needs: if speed is more limiting than quality of the solution, then he will probably choose the low initialization, otherwise, if the best possible quality is needed high initialization may be the choice.

In order to picture the meaning of the resulting Q-values we show examples of how the agent would recover from disturbances. We first look at what happens if the agent - for some reason - finds itself coming home at 4:00 in the morning. Assuming the 15 minutes resolution case, it stays at home until 07:15 and changes then to the next activity which is *work*. From then on the agent is back on his usual daily plan. As another example we want to have a look at what happens if the agent comes to work late. Let us assume that it starts working at 10:00. Again, looking at the Q-values, the agent decides to stay at work for 8 hours (instead of 9.5 hours on a normal day) until 18:00 and then change to the next activity which is *Shop*. Now activity *Shop* starts 30 minutes later than in the optimal case namely at 18:15 instead of 17:45. The agent now decides to spend 30 minutes shopping and to continue then with activity *Leisure*. Arriving at the next activity at 19:00, the agent is still 30 minutes late compared to its optimal daily plan. Then, it spends 4 hours and 45 minutes with leisure activities saving 30 minutes. Finally, the agent arrives at home at 0:15 catching up to its usual daily plan.

The second example reveals what it means to do within day re-planning: the agent chooses a graceful way to get out of the undesired situation. In our case, this is done by gradually saving time where it hurts the least. The agent does *not* try to catch up with its optimal plan at any cost.

3.6 Discussion and Further Work

It is interesting to know what kind of a problem Q-learning is trying to solve. There exist $n_{Q-values} = n_{activities} \cdot n_{actions} \cdot size_{rewardtable}$ Q-values that we are trying to find the steady state for. For the case where the time resolution t_{res} is 15 minutes, $n_{Q-values} = 4 \cdot 2 \cdot 24 \cdot 4 \cdot 12 \cdot 4 = 36864$. Therefore any algorithm will need at least 36864 steps to find the proper Q-values.

But it is to be expected that due to high discount parameters, it will take longer to find the steady state. In order to get a feeling for the problem, let us consider the following fact:

The 10% time horizon - the number of states a reward has to be away from the current state in order to affect it by less than 10% - for a discount parameter of $\beta = 0.99$ is $t_{horizon_{10\%}} = log_{\beta}0.1 \approx 229$, meaning that the states 229 steps (or more than 2 days) in the future from the current state still affect the Q-value of the current state substantially.

If we assume that a fictitious algorithm would know all the optimal paths in the state space in advance (i.e. it would know in each state which action maximizes the cumulative discounted reward) it would need 229 state transitions to calculate the proper Q-value for that state. Since the rewards are transmitted *against* the direction in which the algorithm proceeds, the algorithm needs to take the whole optimal path/cycle 229 times in order to have the reward propagated backwards to below the 10% level. This argument indicates that an estimation of the number of necessary learning steps is $n_{Q-values} \cdot t_{horizon_{10\%}}$, which is $36864 \cdot 299 = 8.6mill$ steps in our case. This is, however, a worst case argument which would, e.g., be fulfilled if all good paths/cycles in the system were completely parallel. In most cases, there will be more interdependence of the Q-values. Also, we are not looking for exact Q-values, but only for good daily plans. In our tests, the algorithm found good plans with only 2 million visits.

After understanding the structure of the problem a bit better, one can in fact envisage much faster algorithms. As becomes clear from the above argument, the slowness of Q-learning lies in the fact that rewards are transmitted against the direction of the algorithm. It turns out that this is in fact only necessary when the expected reward of a state-action-pair is not known (Russell and Norvig, 1995) and the algorithm itself has to do the averaging over the realizations. Since in our case they are known, one can, for example, use faster techniques from Dynamical Programming for the same problem (Karlström, 2004). Alternatively, one could probably even use a generalized shortest path algorithm (Barrett *et al.*, 2000) and simply have it originate at all possible states at a given point in time.

For our implementation of Q-learning, the reward tables need to be filled. These have a large number of values to be defined (for $t_{\text{res}} =$

$15\mathrm{min}$ there are 18432). The question is where to get these from. One option is to use discretized utility functions. For example, one could decide to use logarithmic utility functions, of the type $U_{act}(d) = \alpha log\frac{d}{d_0}$, where d is the duration and α and d_0 are parameters. One could then fill the reward tables by setting the reward for the first 15 minutes to $U_{act}(d = 15\mathrm{min})$, the reward for the second 15 minutes to $U_{act}(d = 30\mathrm{min}) - U_{act}(d = 15\mathrm{min})$, etc. The reward tables would retain the advantage that one could still introduce deviations from those mathematical equations when desired, for example reducing certain rewards for certain times-of-day. In our view, this leads to a very flexible tool.

As mentioned in the introduction, the problem of activity time selection is *too easy* for Q-learning, since it can also be solved by numerical methods. However, the general method of expanding the day into possible states along the time axis is amenable to more complicated formulations. For example, one could allow for skipping activities (i.e. have direct transitions to the second-next activity), one could allow for arbitrary activity sequences, and one could include a limited number of different possible locations for each activity. We therefore believe that the general approach - to see daily activity planning as the question to find good cycles through space-time - provides many interesting avenues for future research.

3.7 Summary

We used the Q-learning algorithm to generate flexible daily activity plans. This was done using reward tables that give the utility per time slot for executing an activity for an additional time slot. This utility per time slot depends on the activity type, the time of day, and the starting time, resulting in complex utility landscapes. The algorithm tries to find an optimal circular path in the activity state space that corresponds to a 24 hours daily plan that can be executed repeatedly on consecutive days.

The solution found is not only an optimal daily plan but it also holds all information necessary to react to unforeseen disturbances. In such a case, the best reaction can be found directly in the table which contains the agent's daily plan.

We applied our algorithm to an example with 4 activities *Work*, *Shop*, *Leisure* and *Home* in order to generate daily plans of different resolutions. With a time granularity of 15 minutes the convergence took no longer than 2 seconds, for 30 minutes temporal resolution the algorithm had to run for approximately 0.5 seconds to find the optimum.

Using this method in a multi agent travel simulation to plan the activities of up to 10 mill agents seems feasible and the resulting within day re-planning capabilities promise computational alternatives to *true* within-day replanning.

Chapter 4

Implementing Activity-Based Models: Accelerating the Replanning Process of Agents Using an Evolution Strategy

Authors

David Charypar
Institute for Transport Planning and Systems, ETH Zurich, Switzerland
Email: charypar@ivt.baug.ethz.ch

Kay W. Axhausen
Institute for Transport Planning and Systems, ETH Zurich, Switzerland
Email: axhausen@ivt.baug.ethz.ch

Kai Nagel
Institute for Land and Sea Transport Systems, TU Berlin, Germany
Email: nagel@vsp.tu-berlin.de

Publication

This paper was presented at a conference, see Charypar *et al.* (2006).

Abstract

We present recent advances in accelerating our agent-based simulation of travel demand by improving various modules of the simulation system. First, the optimization algorithm used in the replanning module is replaced by an evolution strategy that has shown to perform well on a variety of optimization problems including noisy and distorted search spaces. The replanning module is then extended by an accurate way of estimating time-dependent travel times. This makes it possible for the replanning module to produce better plans more quickly. Second, the percentage of computational agents that replan their days is investigated and a percentage that decreases with the iteration number is found to improve the learning speed significantly. On top of these changes a new, fast event-driven microsimulation of traffic flow is incorporated into the model to make the execution of the overall system less time consuming.

Keywords: Daily plan optimization; agent-based travel behavior simulation; evolution strategy

4.1 Introduction

In transport planning, the commonly used aggregated models have limitations in their predictive power as the aggregation process always entails a lack of certain information about the traffic that is predicted. Usually only aggregated values like total traffic volume on a link or average travel times are computed. (Some of these problems can be addressed by using time dependent traffic assignment or disaggregated models.) Consequently, it is interesting to find a transport planning method that is able to predict all aspects of traffic including important information such as distribution of trip purposes of the drivers that cause the congestion on a road or the distribution of income of the people driving on a road during rush hours and separately for off-peak times (or at any second of the day).

Thinking about what such a model could be, we make the following observation: the fully detailed travel demand—including all desirable information about the users of a network—derives naturally from the daily plans of all people traveling in the study area. As a consequence, one way to answer transport planning questions is to find the daily plans for all people interacting in an area.

In order to provide a tool to produce the requested daily plans we are developing the multi-agent travel simulation toolkit, MATSim-T, which is an agent-based microsimulation system of daily demand. The basic idea is to create a synthetic population of agents that live in a virtual world that reflects data as the road network, land use data etc. The agents have daily activity plans that they use to describe how they act in the virtual world. Each agent has the desire to perform optimally according to a utility function that defines what a useful day is[1]. Each agent can change its daily decisions to get a higher overall utility. This can be interpreted as learning. When the agents end up in a situation where none is able to improve his plan they are in a user equilibrium and the learning loop ends. Assuming that a user equilibrium is a state of the system that we are looking for we get a set of daily plans of all agents that represent a typical state of the world.

[1] Note that—at least theroretically—any scoring function would work that defines a cardinal ordering of all daily plans.

Such a learning system can be used for various predictive tasks: for instance, it is possible to obtain the distribution of activity chains of people being at the city center during lunch. Also, the utility of activities can be judged that cause the traffic on certain links. Furthermore, using such a system, road pricing policies can be simulated and their effect can be accurately quantified.

While our current simulation system has proved to work (see for instance Balmer *et al.*, 2006a) there is still a substantial computational effort involved in simulating the learning of daily plans for all virtual persons involved in a scenario. This is especially the case when looking at large scale scenarios with 1 million persons or more.

In this paper, we show how the speed of the overall learning system can be increased by improving the performance of individual modules such as the replanning module (responsible for creating new plans for each agent) and the traffic microsimulation.

The rest of this paper is structured as follows: in Section 4.2 we describe the overall design of our agent-based microsimulation of daily demand; in Section 4.3 we introduce and explain the measures taken to speed up this simulation system; in Section 4.4 we present the test scenario that we use to compare the original and the improved system; in Section 4.5 we discuss our results, and we conclude and give an outlook on future work in Section 4.6.

4.2 The Microsimulation toolkit

The goal of the **M**ulti **A**gent **T**ravel **Sim**ulation **T**oolkit (MATSim-T) is to simulate and predict the daily travel demand of a whole region with one million inhabitants or more. The basic idea of MATSim-T is that by representing each person in such a scenario with an individual agent and simulating the daily behavior of such a person the travel demand is generated as a byproduct.

Technically speaking, MATSim-T divides into several conceptual and computational modules. A simulation *agent* holds several attributes like age, car ownership, home address and suchlike. Additionally, he holds a *daily plan* that represents his activity decisions throughout the day. This information includes an activity pattern with activity types (like home-shop-leisure-home), timing information, i.e. when each activity starts and when it ends, the locations assigned to these activities, and the routes to travel on the trip from one location to another.

When agents decide to change their daily plan, they call the *replanning module* to modify their plan. The replanning module tries to improve the agent's plan. To do so, the utility of a plan is judged using a *utility function* and the task is formulated as a maximization problem. The utility function we use was presented in Charypar and Nagel (2005). It defines the utility of a daily plan to be the sum of individual utilities of each activity present in the plan. This utility of an activity consists of a positive term for the duration that the activity is carried out and negative terms for travel costs and penalties for coming late, leaving too early, etc. Using the idea of penalties, environmental constraints, like shop opening hours, can be handled as well.

Throughout a scenario run, the agents and their plans are held in memory in the so called *agent database*. The interactions between agents happen in the actual *travel microsimulation* where the daily plans of all agents are executed. In the microsimulation, the agents travel from one location to another as intended in their daily plans. As agents have to use the same network, the links in the network become loaded, possibly creating congestion and increasing the travel times on the heavily loaded links. The agents then have the opportunity to change their daily plans to reflect the new situation. The process is iterated and the system relaxes to a point of rest that may be interpreted as a user equilibrium.

As the data format for in- and output to and from our simulation system we use XML. Network, land use, and population data as well as the agent's daily plans are communicated and stored in the XML format. A very important part of our microsimulation toolkit is the initial demand modeling meta-module that comprises a great unified way of reading input data in many different formats, fuse them in a consistent way and store them in a clearly defined format to make it available to our dynamic simulation system. For information about this part of MATSim-T see Balmer *et al.* (2006a)

In this paper we mainly modify three parts of the whole simulation system. First, the replanning module is modified to use a different, more sophisticated optimization algorithm for the search for the optimal daily plan for a given agent under the constraints given by the world and the agent. To exploit the capabilities of this new optimizer, a new, accurate estimation of time of day dependent travel times is included in the replanning module to allow the quality of generated plans to increase. Second, the percentage of the population that actually computes new plans is investigated and optimized for maximal performance of the overall system. Third, a new event-driven microsimulation of traffic flow is implemented.

4.3 Speeding up MATSim-T

An important part of our work is to reduce the time needed for our simulation toolkit to converge to a point of rest at which all the agents are executing a daily plan that they cannot improve significantly themselves. To reduce this time there are a couple of approaches that one can take. One way is to reduce the number of iterations of the whole system that are needed to find a relaxed state. This can be done for instance by improving the replanning module to produce better plans for the share of agents that are replanning. Another way one can go is to find the optimal percentage of agents selected for replanning in a particular iteration. A completely different approach to improve the overall performance of the system is to reduce the time needed to process a single iteration. This can be done by optimizing the individual modules. On overview of the improvements that were done during the work described in this paper can be seen in Figure 4.1.

4.3.1 Producing Better Plans

The first approach to make our simulation system faster is to produce better plans during the replanning process. Again, there were two tracks investigated: first, we tried to improve the optimization algorithm used in the replanning module, second, we made the replanning module aware of the true time of day dependent travel times in the network.

4.3.1.1 A Better Optimizer for the Replanning Module: CMA-ES

So far, the optimization algorithm used in the replanning module of MATSim-T was a specially made genetic algorithm (GA) (see Meister

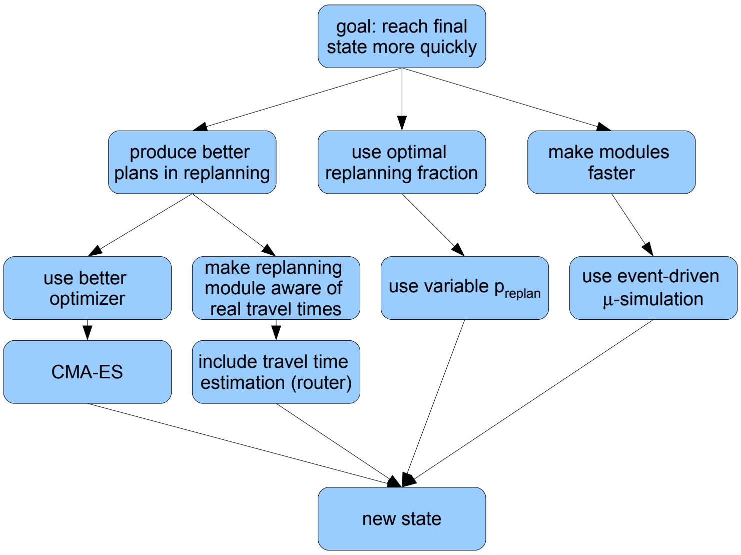

goal: reach final state more quickly
produce better plans in replanning
use optimal replanning fraction
make modules faster
use better optimizer
make replanning module aware of real travel times
use variable p_replan
use event-driven µ-simulation
CMA-ES
include travel time estimation (router)
new state

Figure 4.1: Steps that were implemented to speed up the MATSim toolkit

et al., 2005a). Although its performance was generally sufficient to solve the optimization problem at hand we assume that a more sophisticated algorithm would yield better replanned daily plans or at least provide them with less computational effort. We chose to use the covariance matrix adaptation evolution strategy (CMA-ES) as described in Hansen and Kern (2004) to replace our current GA.

Evolution strategies belong to the field of evolutionary computation. In general, they represent stochastic population based optimization algorithms for continuous space problems that use recombination and mutation operators to produce new candidate solutions.

The particular evolution strategy used in this paper has certain desirable properties which make it attractive for us to use:

- Invariance of order preserving mappings of the objective function
- Invariance to linear mappings of the search space
- Ability to work well on noisy landscapes with steps and ridges
- Has been shown to work well as a global optimizer on many distorted multi-modal test functions

4.3.1.2 Description of CMA-ES

The covariance matrix adaptation evolution strategy (CMA-ES) is an iterative stochastic population based optimization algorithm. It holds a population of candidate solutions where each candidate solution is a point in n-dimensional vector space. n is referred to as problem dimension where each dimension corresponds to an activity duration that has to be planned by an agent. In our case n lies in the range of 2 to 5. CMA-ES also stores a sampling distribution (consisting of a center point and a covariance matrix) that is modified during the optimization process. In every generation a new population of sampling points is generated, the best candidates are selected and they are used to modify the sampling distribution accordingly. Running CMA-ES consists of the following steps:

1. Initialize the center of the sampling distribution (this corresponds to the starting point) and the global step size. Initialize the covariance matrix to an identity matrix.

2. Sample the objective function using N sampling points according to the current sampling distribution (consisting of center, covariance matrix and global step size). N is a parameter of the algorithm and was set to default values according to Hansen and Kern (2004).

3. Evaluate the objective function on all sampling points.

4. Based on these evaluations, select the better half of the sampling points.

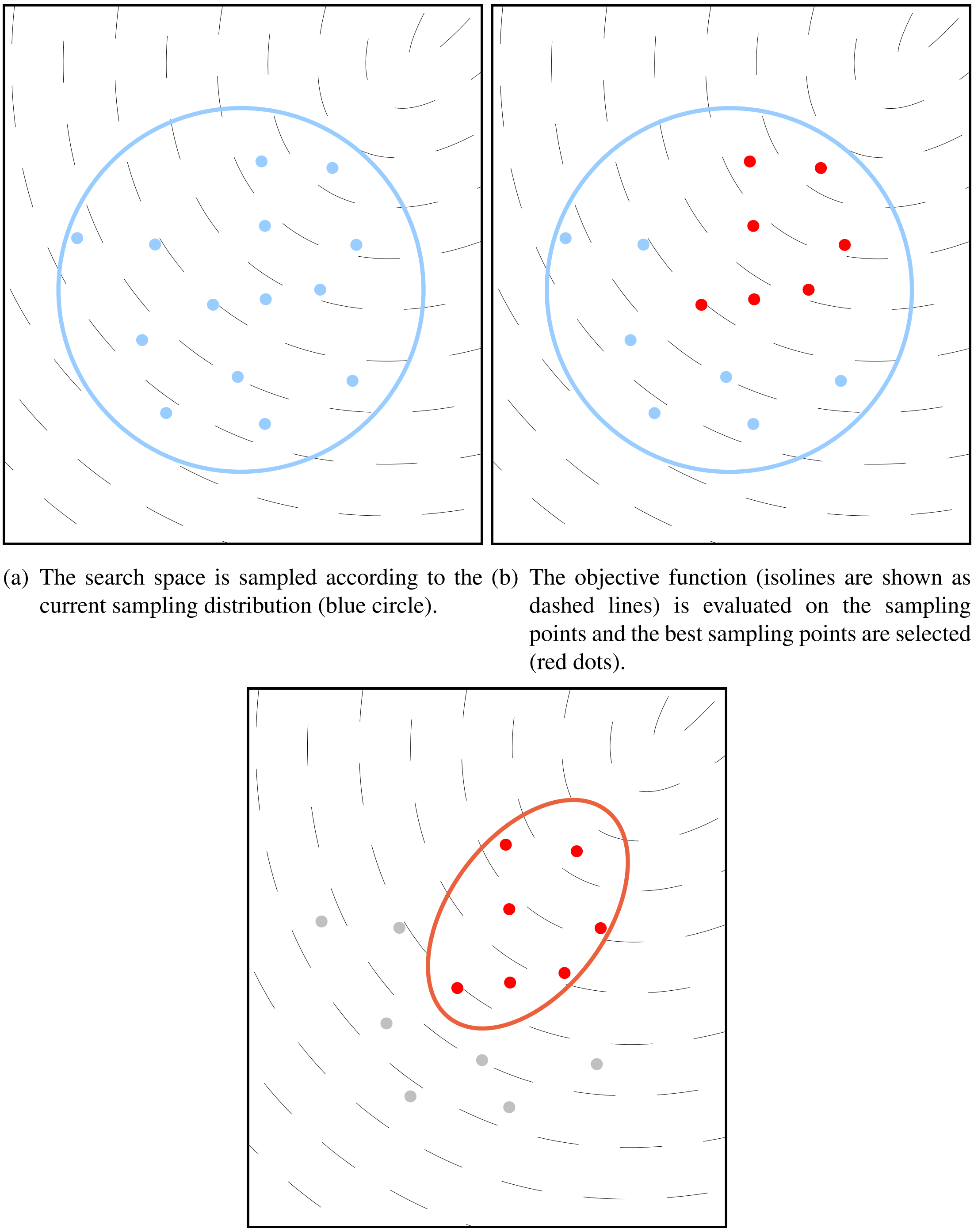
5. Modify the sampling distribution according to the distribution found in the selected samples.

6. Return to step 2 and iterate until a desired stopping criterion is met

For an illustration of the adaptation process of the CMA-ES please see Figure 4.2.

4.3.1.3 Real Travel Times in Replanning

In the state of MATSim-T we were using until now, the replanning module was only in charge of finding a good time allocation for the activities



(a) The search space is sampled according to the current sampling distribution (blue circle).

(b) The objective function (isolines are shown as dashed lines) is evaluated on the sampling points and the best sampling points are selected (red dots).

(c) The sampling distribution is adapted according to the distribution of the selected samples (red ellipse).

Figure 4.2: The adaptation of the sampling distribution in CMA-ES

given in the activity pattern. The travel times were thereby assumed to be constant and equal to the real observed travel times of the corresponding plan executed in the last iteration. This simplistic assumption would of course often prove to be wrong resulting in daily plans performing significantly worse during the execution of a microsimulation run than it was anticipated by the rescheduling module. Strictly speaking, the replanning module was not able to deliver the optimal plan for a given agent as the information on the dynamic state of the road network was simply not available to it. In order to find the real optimal daily plan for an agent the replanning module needed the help of the agent database. This database was holding a couple of daily plans per agent in memory, selecting the plan actually executed during a run according to their respective utility in the last executions. By replacing bad performing daily plans with new plans the agent database would make sure that the whole system would slowly move to better-performing regimes. For more details on the state of MATSim-T as we were using it so far, please refer to Raney (2005) and Balmer *et al.* (2006a).

Although using the agent data base for some aspects of the replanning process has proved to be robust, it also takes very long for the whole system to converge to a reasonable state. As a way around that problem, it seems natural to provide the replanning module with all information necessary to produce the optimal daily plans also in the real world of the microsimulation.

To do so, we include a time of day dependent routing module in the replanning module. For every new candidate plan that is produced while the replanning module runs for a specific agent, the real travel time in the loaded road network is calculated for each of the trips based on their respective departure time. A similar approach is used in Meister *et al.* (2006).

By including an estimation of the real travel times in the replanning module the help of the agent database is no longer required to find good daily plans for our agents. Consequently, the number of plans held in the agent database can be reduced to one.

4.3.2 Using the Optimal Replanning Share

During a run of our simulation system, in every iteration a certain percentage of the population of agents is selected at random to produce new daily plans. The rest of the agents (usually the larger part by far) reuse plans that they came up with in an earlier iteration. This is important to avoid oscillating effects and to make the system converge to the point of rest.

Oscillations can be avoided best by using a very small replanning percentage. Although this is effective in producing a smooth trajectory, this leads to a poor learning speed of the overall system. One would expect that the system would learn more quickly—especially at the beginning—if the replanning percentage would be held at a reasonable high value.

The question is which percentage to choose. What value is the optimal trade-off between a stable solution without oscillations and a quick convergence to the point of rest?

To answer this question we tested the system with different parameter settings for the replanning percentage in the range from $2\% - 50\%$. The results can be seen in Figure 4.3. In good agreement with the reflections above we see that a replanning fraction of 50% produces a chaotic behavior of the simulation system. In the area from 2% to 35%, one can see that higher values lead to a faster learning process and lower values lead to less noise in the solution. For a constant replanning percentage, 5% and 10% seem to be reasonable values.

Judging from the shape of the convergence plots for constant replanning probabilities in Figure 4.3, we decided to use a decreasing replanning probability to get the best of both worlds: a quick learning process at early iterations and the maximal resulting average utility with low noise at the end. After some empirical testing we found that varying the replanning percentage according to following formula produced a significant improvement in the learning performance of the system:

$$p_{\text{replan}} = \min(35\%, \frac{2.0}{n+2}) \tag{4.1}$$

where p_{replan} is the replanning probability and n is the number of iteration. This can be interpreted as an implementation of the method of sucessive averages as described in Sheffi (1985), for instance. The com-

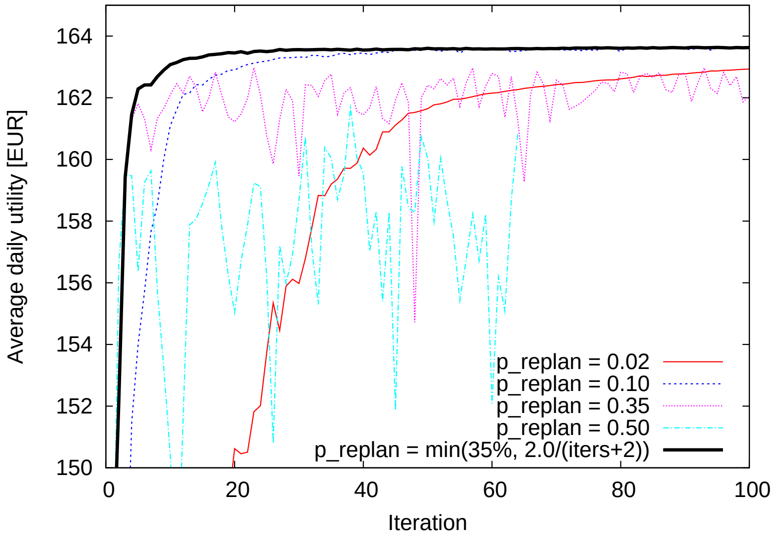


Figure 4.3: Convergence plots for test runs performed to find the optimal replanning share: The system was iterated using various parameter settings for p_{replan}, the replanning probability of an agent in each iteration. The tests were performed for fixed values in the range $2\% - 50\%$ and with a decreasing $p_{\mathrm{replan}} = \min(35\%, \frac{2.0}{n+2})$, where n is the iteration number.

parison of the learning performances with constant and variable replanning probabilities shows that the latter can boost the overall performance of the system by a factor of three or more.

4.3.3 Making Iterations Faster

Probably the simplest approach to improve the performance of our agent-based system is to speed up the execution of the individual modules. If we reduce the amount of time needed for the execution of one iteration, obviously, the time needed to converge to the user equilibrium will reduce as well. We have adopted this approach and worked on the acceleration of the microsimulation of traffic flow that so far represented the most time consuming part of the overall simulation system.

4.3.3.1 The Time-Step-Based Microsimulation of Traffic Flow

For the microsimulation of traffic flow we use queue-based car dynamics on the links. The basic assumption is that especially in urban traffic networks, the behavior of the cars on the roads is mostly dictated by limitations in capacity of the intersections. Cars drive a link all the way down until they reach the end of the queue of cars waiting to cross the next intersection. From that moment on the car has to wait until all the cars in front of it get “served” by the intersection. If a car becomes the first in the queue in every time-step (usually one second) of the simulation it is decided if there is room for it to cross the intersection and to enqueue at the next link (this is only possible if the spill back on the next link is not so large that it already filled up the whole link) and if the capacity constraints on this link allow it to leave the link (only a certain rate of cars is allowed to leave a link). If both criteria are met the car can leave this link and enter the following link; the next car on this link becomes first in queue. It turns out that using these dynamics we do not need precise information about the position of each car on the link. It is sufficient to remember the order in which the cars entered the link and at what time they would reach its end if they could travel at free speed. This kind of microsimulation is very efficient and can be easily run in parallel. For a more detailed look at the microsimulation of traffic flow as we were using it so far see Cetin (2005).

4.3.3.2 The New Event-Driven Microsimulation of Traffic Flow

The microsimulation used until now has two computational issues: first, links that are almost empty still need too much computing time, as even if there is no car waiting at the front of the queue the link has to be checked in every time-step; second, completely congested links also absorb a fair amount of computing time as the simulation has to check in every time-step if it is possible for the first car in the row to cross the intersection. By looking at how to improve the speed of the present implementation of the traffic microsimulation, one idea is to try to put the computing time where the action is: simulate only where traffic is currently happening, i.e. where cars enter and leave links. Consequently, empty links and cars waiting on a link should not need any computing time at all. Completely congested links should need almost as little computing time as empty links as the cars do not move on such links. One way to achieve these requirements is to use an event-driven microsimulation. In our approach, we use timers that are set by the agents for the time that they plan to enter or leave the links. These times are estimated by collaboration of the agent and the links involved. Our approach is similar to the work presented in Marchal (2001) but in contrast, our simulation is fully microscopic also on the supply side. Another event-driven simulation was used for the simultaneous simulation of activity chains in Axhausen (1988).

A detailed discussion of the algorithms and concepts used in our new microsimulation are beyond the scope of this paper but we would like to describe an example situation and how computing time is saved in our approach. Assume a car C traveling on link A and wanting to enter link B. Let link B be completely congested and therefore, when the agent reaches the end of link A, it cannot enter link B directly. Instead it *registers* his desire to enter with link B. As long as no car leaves link B nothing is going to happen and consequently no more computing time is needed to handle the situation. But, as soon as—for any reason not described here—a car leaves link B this creates a cascade of events. A new *gap* is created at the front of link B. This gap travels backward through the link at a fixed speed. As a consequence of the fixed speed, link B is able to predict when the gap is going to reach the entry point of link B. As car C is still registered with link B, this predicted time is communicated to car C. Following this, the car C is going to create a timer for this time and when it expires car C is going to leave link A and enter link B. Note that computing time was only used for (1) registering to link B, (2) computing the gap arrival time, (3) registering a timer, (4) leaving link A, (5) entering link B. Apart from step (2), there is no overhead due to the fact that the link is congested, or that there are many cars interacting on the links.

It is difficult to compare our old microsimulation with the new event-driven approach as the computing time depends on the scenario simulated as well. But our first test show that for 24 hours simulations of scenarios with about 1 million trips we gain roughly a factor of ten in terms of computing time.

4.4 Test Scenario

For all our testing we use a scenario of the canton of Zurich. The population of agents was taken from Frick and Axhausen (2004) but the number of agents was reduced from 550 808 to 12 225 using a sampling process. This was done to make quick test runs possible even on single CPU main stream desktop computers. The network capacities were reduced accordingly to produce a fair amount of congestion in the network and make the learning task more demanding. We believe that this setup shows similar characteristics as the full size scenario. However, we want to test and show the performance of our simulation system on the full-sized problem in the future as well.

The network we use has approximately 20k links and covers all of Switzerland. While it is possible for the agents to travel the whole network (e.g. an agent could theoretically decide to drive from one location in the canton of Zurich to another location in Zurich via Geneva, 300 kilometers away from Zurich) the part of the network that is used most of the time only comprises roughly 4000 links. Figure 4.4 shows an extract of the network of the city of Zurich, Switzerland. The image was produced using the MATSim-T visualization tool.

During initial demand modeling, the activity chains as well as the locations for these activities are generated according to micro census data, a commuter matrix, and land use data. For details please refer to Balmer *et al.* (2006a). The resulting initial demand for the learning loop of our agent-based system is shown in Figure 4.5. Note that it shows some observable properties of normal urban traffic like a morning and an evening rush hour but the the plans are very undifferentiated and certain properties are quite far from reality such as the evening peak that starts already at 14:00.

The average number of car trips that is carried out by each agent is 2.24 and the average trip length in number of links traveled is 6.94 after convergence to the user equilibrium. While the average number of car trips is too low for a realistic scenario this does not represent a limitation of our model but is due to the fact that the initial demand used was originally designed for a different application and already shows this property.

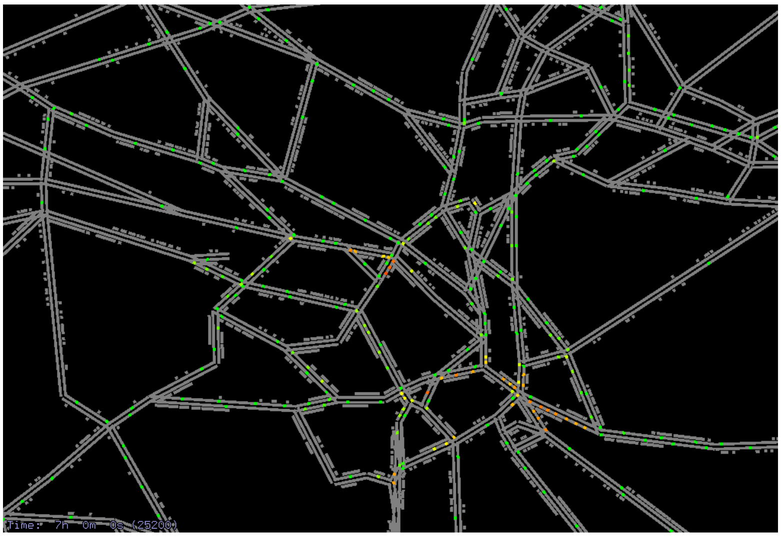


Figure 4.4: Visualization of a scenario run using the MATSim-T visualization tool. Shown: the city of Zurich, Switzerland. The colored dots are cars driving through the network. Green, yellow, and red cars travel at free, half, and low speed, respectively.

4.5 Results

We compare the learning performance of our agent-based simulation system after incorporating the changes described in this paper to the state before. To do so, we make a comparison based on the average utility of the executed daily plans of the agents as well as a comparison of the load profiles of the network during the day.

4.5.1 Comparison of Learning Performance Based on Average Utility of Plans

To measure how well the agents in our simulation have learned to perform their activities and to adapt to the system under load we use the average utility of their plans as they were executed in the microsimulation module. In the following plots, higher values mean a better adaptation to the problem at hand. However, higher average values do not always depict a better overall performance. One has to keep in mind that we are looking for a user equilibrium while the average score would be at

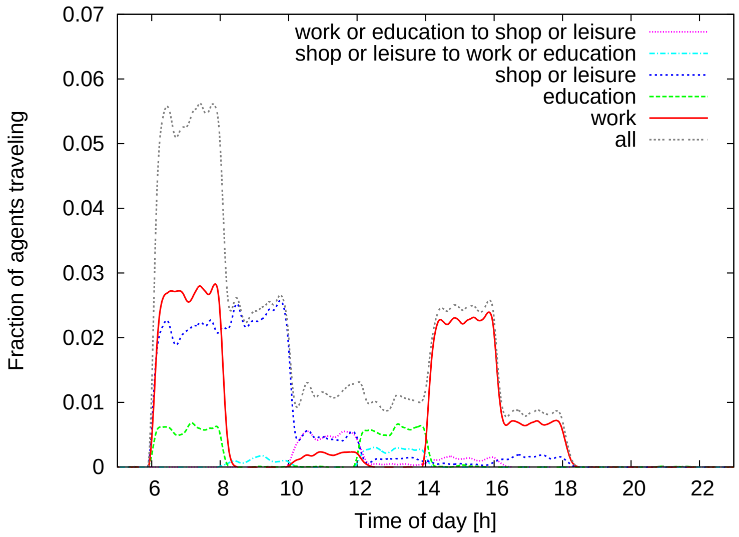


Figure 4.5: Network load according to initial demand. This plot shows the network load according to the initial daily plans of the agents before iteration 0. Note that the demand cannot be executed in this way. The plans of the agents assume an empty road network and corresponding travel times. The real execution leads to a very congested network and therefore to a big discrepancy between the planned daily plans and the executed ones.

a maximum in the system optimum; these two are not necessarily the same. Unfortunately, it is not easily possible to find out if the agents are in a (stable) user equilibrium and therefore we nevertheless use the average score to judge the performance of our simulation system.

In Figure 4.6, a comparison of our simulation system before any changes and the current state can be seen. The system in the new state learns substantially faster than the original system. The modified system reaches the stable plateau close to the maximum after approximately 20 iterations. The original system shows a local maximum at about 70 iterations but continues to rise after iteration 100 and reaches a similar average utility after approximately 900 iterations (not shown). If we take iteration 70 as final result of the original setup this means that our new system is a factor of 3.5 faster in terms of iterations.

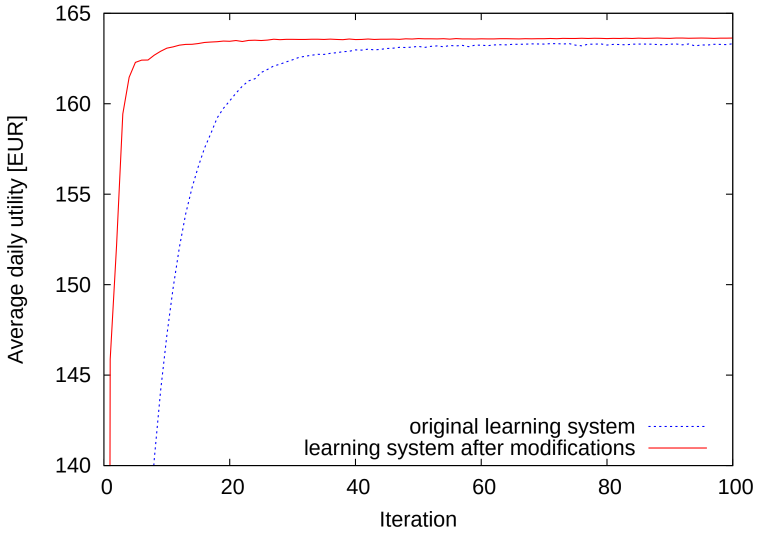


Figure 4.6: The modified simulation system learns substantially faster than in the original state.

4.5.2 Comparison of Results Using Network Loads

We are interested in the behavior of the agents after learning. In particular we want to know when they travel and for what purpose. To investigate these questions we plot the time dependent network load by purpose.

In Figure 4.7, we see a fictitious network load as it is planned by the agents if they assume infinite network capacities and free speed travel times. High values in this plot mean that many agents are traveling at the same time. This plot can be used as an indication of how the agents would plan their days if there was a guarantee of no congestion at any time. Note the pointed morning and evening peaks and the relatively flat distribution of shop and leisure trips throughout the day. It is interesting to compare this “ideal” demand to the real executed demand shown in Figure 4.8 using our new learning system after 20 iterations. It can be seen that the morning and evening peaks are significantly broader (the morning peak starts earlier and the evening peak ends later) and that people on shopping and leisure trips systematically avoid these peak hours for their travel. We conclude that our learning system is able to reproduce a typical behavior of people in daily life: they avoid peak hours for travel to and from activities that are not bound to a specific time.

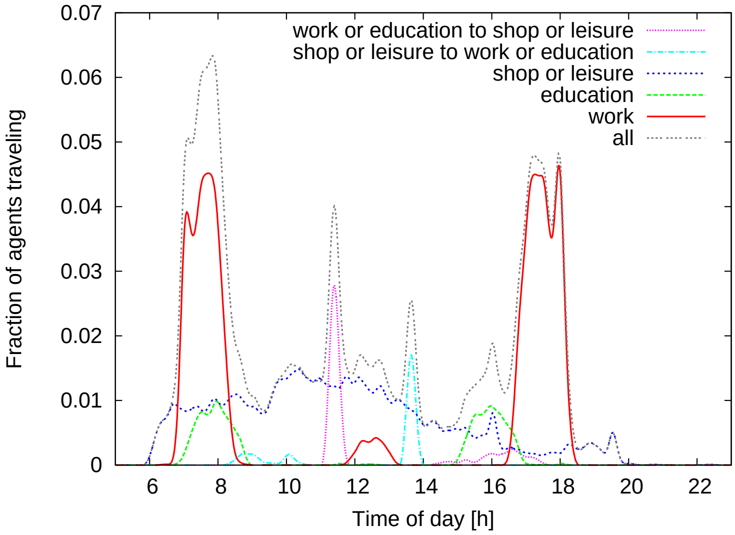


Figure 4.7: Optimal demand assuming free speed on all links. The demand as it is planned if the agents assume free speed travel times on all links. The morning and evening peaks present are very pointed. Note that shop and leisure activities happen throughout the day and that they do *not* interact with the rush hours. Assuming free speed there is no need to avoid heavy traffic. It is clear that this demand cannot be executed as planned, as the network would become totally overloaded.

In Figure 4.9, we show the same plot as above for the original, unmodified learning system after 70 iterations. The peak hours look very similar to Figure 4.8, but no interaction effects between shopping and leisure activities and the peak hours can be identified. This shows that the the original system needs significantly more time to reach a certain solution quality. After further investigation we found that in iteration 200 the old system already has developed the avoidance effects partially, but that it takes roughly 400 iterations before a similar structure can be observed. The new system therefore needs 20 times less iterations to reach the final solution quality than the old system.

Going into the other direction, and looking at the resulting network load after 10 iterations of the new, modified learning system we found that while not being as far as after 20 iterations the network load looks already very similar. Especially the described drop of shop/leisure demand can be clearly identified.

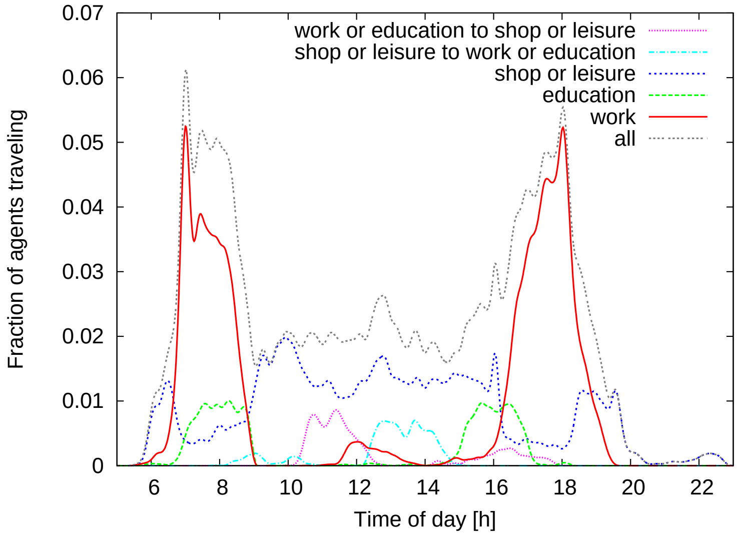


Figure 4.8: Demand after 20 iterations of learning of the new simulation system: The agents have learned to avoid the morning and evening peaks if possible. These peaks are still very dominant but significantly broader than in the "free speed" demand. Note how shopping and leisure trips are carried out during off-peak hours to avoid congested roads.

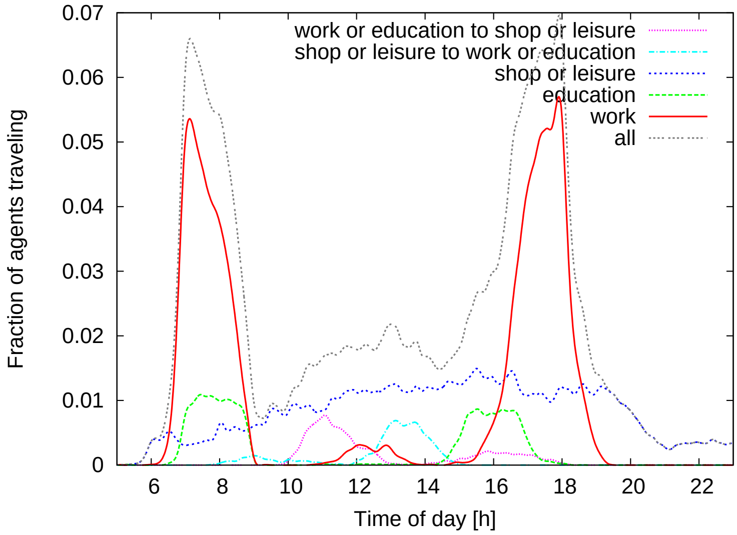


Figure 4.9: Demand after 70 iterations of learning of the original learning system: While the morning and evening peaks have developed, shopping and leisure trips do no yet avoid the peak hours.

4.6 Conclusion and Outlook

We have improved our agent-based iterative microsimulation of daily travel demand by enhancing its replanning module in two ways: first, an optimization algorithm that was shown to perform well on relevant test problems (the covariance matrix adaptation evolution strategy CMA-ES) was integrated; second, the power of this optimizer was exploited to produce better plans for the agents by including an accurate travel time estimation (a time-dependent routing module) into the evaluation function for daily plans. This makes it possible that the replanning module directly reacts to peak hours and their high cost of travel. These modifications have the positive effect that they significantly reduce memory needs as they eliminate the need for the agent database (another module of our simulation system) to hold multiple plans in memory.

We have tested our new agent-based model with a reduced test scenario with 12 225 agents and have shown that these modifications yield an improvement of a factor of 20 in terms of iterations needed for the agents to adapt to the network under load.

On top of the other modifications, a new event-driven microsimulation of traffic flow was implemented and integrated into the system to increase the speed of execution of a single iteration. This improves the speed of execution of the microsimulation of traffic flow—a subtask of one full iteration—by a factor of 10.

The current replanning module is only able to create optimal time allocations given a location choice decision and an activity chain. In the future we would like to remove this limitation and enhance our replanning module to incorporate the whole replanning problem.

Chapter 5

An Event-Driven Queue-Based Traffic Flow Microsimulation

Authors

David Charypar
Institute for Transport Planning and Systems, ETH Zurich, Switzerland
Email: charypar@ivt.baug.ethz.ch

Kay W. Axhausen
Institute for Transport Planning and Systems, ETH Zurich, Switzerland
Email: axhausen@ivt.baug.ethz.ch

Kai Nagel
Institute for Land and Sea Transport Systems, TU Berlin, Germany
Email: nagel@vsp.tu-berlin.de

Publication

This paper was presented at a conference (Charypar *et al.*, 2007b) and later published in a journal (Charypar *et al.*, 2007c).

Abstract

Simulating traffic flow is an important problem in transport planning. The most popular simulation approaches for large scale scenarios today are aggregated models. Unfortunately, these models lack temporal and spatial resolution. On the other hand, microsimulations are very interesting for traffic flow simulation as they can accurately simulate features requiring both high temporal and spatial resolution, including traffic jams and peak hours. However, most microscopic approaches involve high computational costs, requiring expensive large computers to run them within reasonable time. In this work, we present possibilities for reducing these costs by using a queue-based model and an event-driven approach jointly. Our approach makes it possible to run large scale scenarios with more than one million simulated person-days on networks with 10k links in less than ten minutes on single CPU desktop computers present in most offices today.

5.1 Introduction

Traffic flow simulation is an important problem in transport planning, as it represents the final link between the intangible description of travel demand and the emergence of flow densities, volumes, and travel speeds. In today's practice, traffic flow is most often simulated using aggregated models that are easy to use and well established in the community. However, compared to aggregated models, the traffic flow microsimulation has certain clear advantages:

- Very high spatial and temporal resolution. Features like traffic jams and rush hours can be captured much more accurately.
- Traffic is represented in a very natural way; vehicles traveling, roads, and intersections are simulated directly.
- Traffic flow microsimulations can be easily coupled with other microscopic approaches, i.e. agent-based demand modeling.
- Inverse analyses can be carried out to determine where certain cars are coming from and why they can be found at a specific road network location.

However, these properties come at the price of high computational burden. This very often makes it necessary to use complex software architectures together with large parallel computers - an expensive process. The other option is to use microsimulations only for small applications and switch to the above-mentioned aggregated methods for large scale problems. Unfortunately, these methods are so different that it is nearly impossible to transfer know-how from one field to another.

This work aims to present a new event-driven queue-based approach that makes the traffic flow microsimulation of large-scale problems feasible on affordable, desktop computer hardware within reasonable time.

5.2 Related Work

In this section, we present a selection of previous work related to our microsimulation. There are many different traffic flow simulation approaches. The physically based microsimulations (e.g. AIMSUN (Barceló *et al.*, 1998; AIMSUN, 2006), MITSIM (Yang, 1997; MITSIM, 2006), VISSIM (VISSIM, 2006)) generally try to capture as many traffic flow phenomena as possible. They simulate car following and lane changing behavior and use a continuous representation of space and constant very small time-steps to simulate cars on the roads.

A different microscopic, but less physical, simulation approach is represented by cellular automata (e.g Brilon and Wu, 1998), used for example in TRANSIMS (Nagel *et al.*, 1998; Nagel and Rickert, 2001; TRANSIMS, 2006). Here, cars move through cells that they can occupy like particles. Although we have a coarser level of detail in cellular automata, features like densities and travel speeds still emerge from the cars' simple direct interactions and are not computed at an aggregated level. This changes when we move on to mesoscopic modeling. In mesoscopic models (e.g. METROPOLIS (Marchal, 2001; de Palma and Marchal, 2002), DynaMIT (Ben-Akiva *et al.*, 1998; DynaMIT, 2006), DYNASMART (Chang *et al.*, 1994; DYNASMART, 2006), DYNEMO (Schwerdtfeger, 1984), ORIENT/RV (Axhausen, 1988)) where vehicles are still represented microscopically, travel times and speeds are calculated using aggregates.

The highest level of abstraction can be found in macroscopic models that compute all traffic quantities on an aggregated level. One example of a traffic simulation model as a one-dimensional incompressible fluid is NETCELL (Cayford *et al.*, 1997).

The work presented here builds on the approach of MATSim-T (Cetin, 2005; MATSim-T, 2006). While using simplified, queue-based dynamics to be computationally efficient, this model still estimates all quantities microscopically, and thus should be located somewhere between mesoscopic approaches and cellular automata. The same holds for the approach presented in Mahut (2000) where the spatial resolution on the links is minimized by only calculating the entry and exit times of vehicles.

Time-step based approaches have been more popular in the past than event-driven approaches. It is not clear why, but one reason might be the more straightforward implementation of time-step based simulations.

5.3 Classification of Traffic Flow Microsimulation Methods

5.3.1 Cellular Automata

Cellular automata are used in traffic flow microsimulation to accurately model the behavior of cars traveling on a road network (see e.g. Chowdhury *et al.*, 2000). The basic idea is to discretize space in cells of equal size, each of which can be occupied by, at most, one vehicle. The cars drive through these cells by choosing their speed according to the space available in front of them. Cellular automata have the advantage of simplicity while still retaining a fair amount of spatial resolution and therefore the general ability to simulate many interesting phenomena like car following and lane changing as demonstrated in Brilon *et al.* (1998); Nagel *et al.* (1998). An additional advantage is that link capacities are generated from the properties of the links and the behavior of the drivers.

Although the above properties are generally desirable, the major drawback of this method is its computational cost. Its high temporal and spatial resolution is achieved by granting processor time to every simulated agent in every simulated time-step (usually one second). This becomes impractical if the number of simulated agents rises above roughly 100k, especially if one wishes to simulate the traffic flow repeatedly; perhaps as part of an iteration search for a steady-state solution. If we want to stick with this kind of solution, the only way to achieve sufficient speed for large problem sizes is to use massively parallel computers as illustrated in TRANSIMS (2006). Adequate funding is an ongoing problem for this expensive approach.

5.3.2 Queue-Based Simulations

If the computational speed of cellular automata is not sufficient for a certain kind of application, one way of achieving higher performance is to simplify the model by using a queue-based approach as it is used in MATSIM-T, see Cetin (2005); MATSim-T (2006). Here, the basic assumption is that the main features of (at least) urban traffic can be modeled by looking at the intersections alone. This assumption is based on the observation that in urban traffic networks either: 1. traffic is flowing more or less freely on the links, or 2. the cars are queuing in front of the next intersection and waiting for the car in front of them to move. Based on these assumptions, cellular automata can be modified as follows:

- Cars traveling through the network are no longer simulated directly; instead we change perspective and simulate the links between intersections.
- A queue stores the cars that enter each link and the entry time for each car is stored in this queue.
- The link's capacity and space available for cars are given as parameters.
- When a car wants to leave link A to enter link B in the next simulation time-step, link A checks whether the car is able to leave the link by monitoring various constraints (capacity, free speed travel time, intersection precedence, and space available at the next link). If all preconditions are met, the car leaves link A and enters link B.

An advantage of this model is the reduction in amount of information processed by not having to compute the fine-grained stop and go interactions between following cars. As the basic simulation unit is the road segment, not the car, we gain approximately a 10 to 100 factor compared to cellular automata.

5.3.3 Event-Driven Queue-Based Simulations

Although queue-based simulations are much faster than cellular automata, it is still a computing time challenge to simulate the traffic flow of a whole region microscopically. One seeks even faster ways of computing the same information. One key element is the inefficiency of queue-based simulations in areas where flow density is very low. Here, the links have to be simulated in every time-step, even when there is little data to compute: in most of the time-steps, the links only realize that there is no car in the position to leave the link and enter the next. These empty operations cost the simulation most computing time when simulating off-peak and night hours. One of the goals is thus to speed up low-density traffic simulation.

In order to achieve higher computational efficiency, we chose to substitute constant time-step for the direct treatment of actions occurring on the road network. Each such action is reflected by an event. Every time a car enters or leaves a link, a corresponding event is processed. This means that the number of events happening in a specific time period directly corresponds to the amount of traffic flow present on the network. The computational effort is then proportional to the traffic load. The two main advantages of the event-driven approach are:

- Processing time is assigned according to the traffic flow during any time of the day. As a result, most computational time is spent where traffic flow is maximal and almost no time is spent where the traffic network is empty.

- The total computing time needed for simulation of a certain travel demand (e.g. 1 million cars traveling from one part of a city to another) is almost independent of the total amount of simulated time needed to process the traffic demand of the scenario. In other words, it does not matter if the cars travel during 24 hours or one month. This is because the simulation cares only about the events to be simulated; the total number of events in both scenarios is the same.

When switching to an event-driven simulation, overhead costs (added in situations where the system processes large numbers of events at one time) might be a concern. These are the situations where time-step-based approaches are usually very efficient. Such situations may well arise in certain scenarios, and for these, the queue-based approach with constant time-steps is the most efficient choice. However, such a scenario would need to have full network load nearly everywhere. Our experience shows that, in most simulation networks, there is a fair number of links that experience little load even during rush hours. This might be the reason that we have not yet found a scenario where the event driven approach is slower than the time-step-based approach, even during the most heavily loaded minute of the day. In our tests, the event-driven simulation obtains an overall speedup factor of 20 or more over the time-step-based implementation when simulating a 24-hour scenario.

In METROPOLIS (see de Palma and Marchal, 2002), a similar, event-based approach is used. The important difference there is that

	Spatial discretization	Temporal discretization
Cellular Automata	equidistant	constant time-step
Queue-based simulation	road segments	constant time-step
Event-driven, queue-based simulation	road segments	adaptive, event-driven

Table 5.1: Discretization schema of different microsimulation approaches

travel times and speeds are estimated using aggregated values, while in our implementation no aggregates are used.

A comparison of the simulation approaches' spatial and temporal discretization schemas is shown in Table 5.1.

5.4 Improved Dynamics of the Event-Driven Queue-Based Approach

Apart from the changes that were made to our model to switch from fixed time-steps to event-driven information processing, we have also incorporated several behavioral changes to the road segment dynamics. We want to give an overview of these and briefly discuss the reasons for the modifications.

- *Gaps, gap travel times*: In the classical queue-based approach used until now in MATSIM-T (see Chowdhury *et al.*, 2000), there is no concept of gaps between cars. Assume a full road segment. If the first car in the row is able to leave the road in one time-step, the space available on the link will become visible upstream at the entry of the link exactly one time-step later. That is, congestion can dissolve faster than usually observed because the backward traveling speed of gaps is very high. In order to avoid that problem, we have introduced a concept of backward-traveling gaps on the road segments. These gaps travel in the opposite direction of the cars at a predefined and parameterized speed. In order for a car to enter, there needs to be a gap available at the inlet of a road segment. Otherwise, the car needs to wait for the next gap to arrive. In this way, it is possible to reproduce the typical behavior of backward-traveling gaps on links during unloading.

- *Capacity constraints at inlet*: In the classical queue-based approach, link capacities were only enforced at the downstream end of links. While this is certainly sufficient to reproduce the correct capacities on a linear sequence of road segments, more complex behavior can be observed at intersections: especially on converging Y-type connections with high capacity incoming roads and a relatively low capacity outgoing road, one would expect to observe breakdowns at the bottleneck. However, without limiting the incoming capacity of a road segment, the breakdown does not occur until much later downstream. To avoid this problem, we have extended our model to account for the inflow capacity as well. Note, however, that it is often not easy to get the corresponding data. Setting incoming and outgoing capacities identically is usually an unsatisfactory choice because real incoming capacity is often close to the theoretical capacity of a link of a certain type while the outgoing capacity is often chosen taking traffic light timings into account.

- *Handling of gridlock*: One issue, not very apparent at first, for all types of microscopic traffic flow simulations with predefined routes, is gridlock. In heavily loaded road networks, it is possible that individual agents block each other when waiting for the next link on their individual routes to become free. At a certain point, this will result in gridlock, preventing the involved agents from arriving at their desired locations. Thus, we must incorporate some techniques for handling gridlock into the simulation model. In our simulation, the problem is solved using a guaranteed minimum capacity for each link. This can be, for instance, one percent of the nominal capacity or any constant desired. The simulation then simply guarantees that no matter whether a road segment is already full or not, it will always accept incoming cars - at least at this specified minimum rate. This can lead to overfilled links, but effectively avoids a situation where cars stay in the simulation forever. In our experiments, this method has proven to be practical and quite simple to implement.

5.5 Software Design: Technical Description

In the following section, we present the different parts of our event-driven queue-based traffic flow microsimulation and we will explain how these parts work together to solve the simulation problem at hand.

5.5.1 Description of Travel Demand

Travel demand is the input for any traffic flow microsimulation and it is therefore critical to find a consistent description for it. In this work, we employ the same formalism used in Balmer *et al.* (2006b):

- We use a population of agents, each having a complete 24-hour activity plan.
- An activity plan consists of an activity pattern describing the type and order of activities executed, location choice information describing the activity in question should be executed, timing information defining the time of day when the activities are taking place, and routing information describing the sequence of road segments taken to travel between subsequent activities.

Note that while we use a complete description of traffic demand, the description of the activity type is not necessary for the traffic flow microsimulation.

5.5.2 The Traffic Flow Model

The basic concept of our traffic flow model is that we only simulate transitions of agents from one road segment to the next. During the time spent on a specific road segment, very little information about the position of the agent is available: only the position in the queue is stored and the earliest time that the car could leave the link according to free speed limitations. When the agent arrives at the end of a road segment, an event occurs and the agent leaves the road segment and enters the subsequent one according to the route stored in his complete activity plan. However, this can only happen if there is sufficient room left on the downstream link to take the oncoming agent. Otherwise, the agent has to wait until sufficient space is available.

Travel times derive from the behavior of the agents on the link: if a link is empty, the travel time equals the time needed to drive down the link at free speed. The travel time can lengthen if there are other agents traveling in front of the agent. In this case, the agent cannot leave the link before all agents in front of it have left the link. The reason for such a delay is usually high demand coupled with limited outflow due to limited outflow capacities or a blocked road network downstream. That means, as expected, high demand and low supply will automatically lead to long travel times.

5.5.3 Simulation Output

The primary output of our traffic flow microsimulation is a log-file that lists each event during the course of the simulation run. The complete description of an event consists of the absolute time of day when the event happened, an agent-ID and a link-ID specifying which agent was involved and where the event happened, respectively, and the type of event e.g. entry or exit. Using simple post processing, this *events-file* can be converted into count data, hourly volumes on links, person diaries, or data similar to GPS floating-car data

5.5.4 Actors During the Course of the Simulation

In our simulation software, three basic elements combine to compute traffic flow through the network (see Figure 5.1). These three elements are:

- The road segment that holds the cars during their travel.

- The agent that 1. represents the object moving through the road network and 2. stores all information about the travel demand.

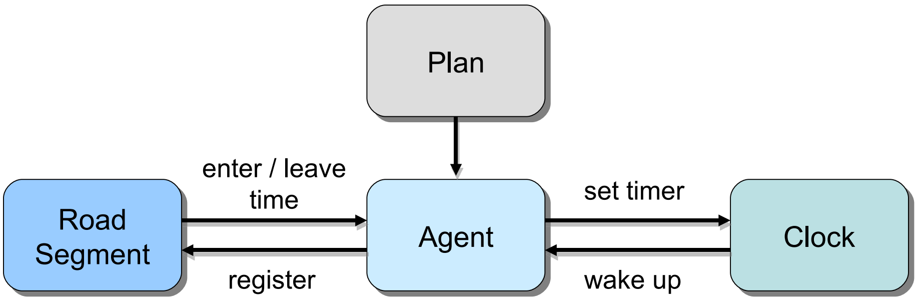


Figure 5.1: Interaction processes between elements of the traffic flow microsimulation

- The clock that handles registered timers and alerts the agents when an action must be undertaken.

5.5.5 The Traffic Flow Model

The actors mentioned above all interact and communicate according to a certain protocol explained and illustrated by the example in Figure 5.2, where one agent wants to travel along a certain sequence of road segments. The protocol consists of the following steps:

1. The agent informs the road segment that it wants to enter. The road segment stores this request chronologically together with all other requests of this kind.

2. As soon as it is clear when a gap will be available for the agent, the road segment sends this entry time to the agent.

3. The agent registers a timer for this time with the clock.

4. As soon as the timer expires, the clock sends a wake-up call to the agent and informs it about the time.

5. The agent then enters the road segment. With this step, the first part of the protocol finishes, and the agent continues to travel down the road. Note that this does not require any actions to be taken by the road segment, the agent, or the clock.

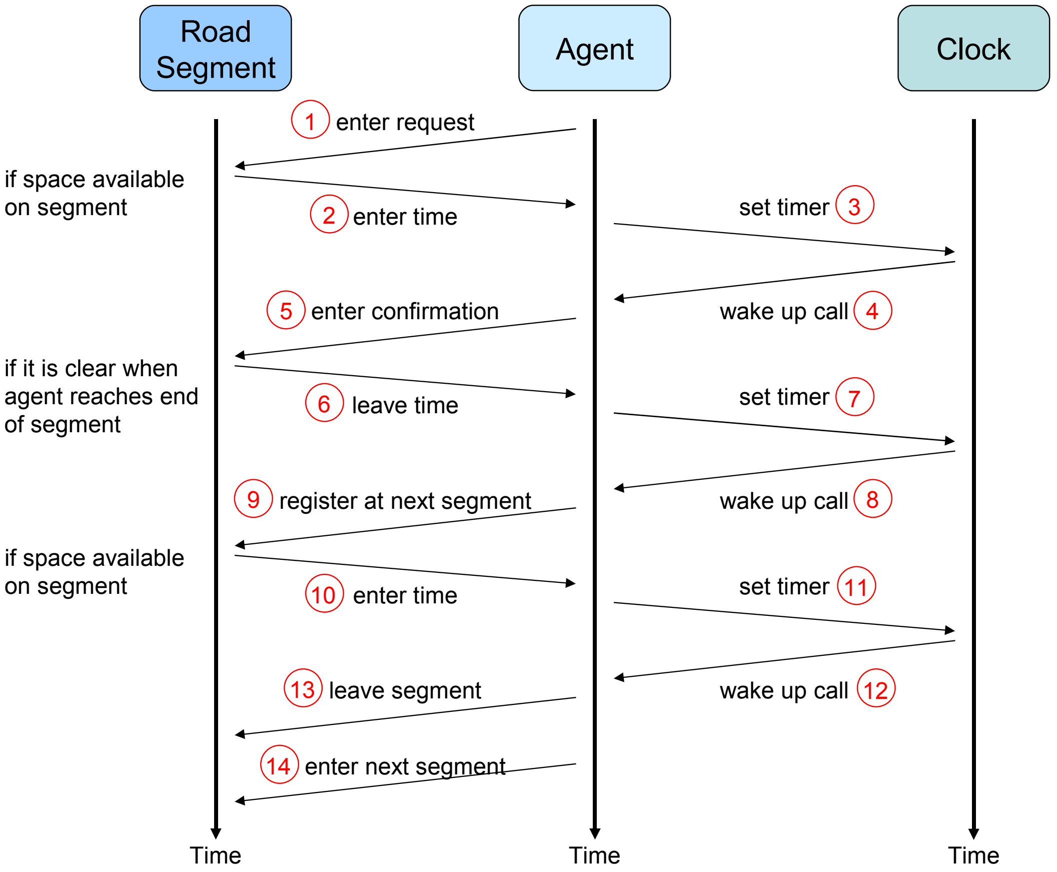


Figure 5.2: The traffic flow protocol

6. When the agent moves up to first place in the road queue, the road segment computes the time the agent is going to arrive at the end of the road segment and sends this information to the agent.

7. Similar to step 3, the agent registers a timer with the clock.

8. After the timer expires, the clock calls the agent.

9. Now the agent is at the front of the road segment. It then checks with its activity plan for the next road segment to travel and registers there.

10. When it is clear at what time there will be a gap available for the agent to enter the next link, it receives a message with the predicted entry time.

11. The agent again registers a timer for this time.

12. When the timer expires, the clock wakes the agent.

13. The agent informs the last road segment that it is leaving.

14. It also informs the next road segment that it enters. From here on, steps 6 to 14 repeat until the last road segment of the trip is reached; then, step 14 is left out and the protocol stops.

5.6 Results

5.6.1 Test Scenario

The test scenario we use to demonstrate the properties of our simulation software consists of the following parts:

- The road network of the federal states of Germany Berlin and Brandenburg. It consists of 11.6k nodes and 27.7k links.
- The synthetic population of the area consists of 7.05M people. Each person has a complete daily activity plan with multiple activities and trips. That means we are simulating 7.05M person-days.

The average number of trips per agent in our demand is 2.02 and the average length of a trip is 17.5 links. This leaves us with an overall daily demand of 249M road segments to be traveled.

5.6.2 Test Setup

The test scenario was run on a server-system equipped with Dual Core AMD Opteron Processors 275 with 2.2GHz. Only one core was used during our test runs. The system was equipped with 4GiB of RAM of which roughly 3.2GiB were used. The input file for the synthetic population and the daily demand uses 5.4GiB of disk space; the output file generated, containing each event processed during the simulation, consists of roughly 17GiB. Note that one reason for the considerable sizes of our input and output files is the fact that we are using ASCII representations (XML in the case of the input files and line-based text files for the output).

5.6.3 Test Results

We ran the test scenario on the machine mentioned above. An average run took an overall execution time of 53 minutes. This time divides into 8 minutes for the reading in and parsing of the input data (network, synthetic population, and complete daily plans), 37 minutes for the execution of the plans (i.e. the actual traffic flow microsimulation), and roughly 8 minutes for the generation of the output files. Summing everything up, this results in an overall real-time ratio of 27 for our test scenario on the hardware described. This means that our simulation ran 27 times faster than real-time. Compared to our original queue-based microsimulation using fixed time-steps, we gained a speedup of more than a factor of 20. Please note that the time needed for I/O could be easily reduced to about two minutes by switching to a binary representation of the input and output files.

5.7 Discussion and Outlook

An interesting question is how to derive a performance estimate for the current microsimulation of the problem to be computed. Fortunately, this is easy because the main factor affecting the computational effort needed to simulate the traffic flow is the overall traffic volume. Judging from our test results, it can be estimated that the average simulation speed on the computer hardware described above is about 110k road segments per second (not including I/O). This can be transferred into a runtime estimate for a given problem: for instance, let us assume a large-scale traffic simulation problem with a population of one million agents, each executing 3 trips a day. Let the average trip length be 15 road segments. The overall traffic volume is then given as $V = a{\cdot}t{\cdot}l = 1\text{M}{\cdot}3{\cdot}15 = 45\text{M}$, where V is the total traffic volume, a is the number of agents in the population, t is the average number of trips per agent, and l is the average length of a trip in number of road segments. From $r = V/v$, where r is the expected running time and v is the simulation speed in number of traveled road segments per second, it follows that $r = 45\mathrm{M}/110\mathrm{ks}^{-1}$. We therefore expect that such a simulation will take about seven minutes to complete on today's computing hardware.

It is interesting to note that the dominant factor limiting scenario sizes when using our microsimulation model is computer memory, not processor speed. Even though the large scenario used for our testing can be simulated in only 53 minutes on affordable hardware, it needs roughly 3.2GiB of memory, meaning that it is impossible to double the scenario size on a machine with 4GiB. This is not an actual problem with our implementation, but arises from the sheer volume of the data describing demand. It would be interesting to investigate how much the memory requirements can be alleviated by using some sort of streaming of travel demand: an agent could postpone loading its trips into memory until just before they are executed. Another possibility might be using some sort of compression algorithm to reduce the memory size needed.

A completely different way around the problem of memory sizes is parallelization. We are developing a parallel version of our microsimulation model that will make it possible to take advantage of a group of cheap computers to simulate large scenarios that would not fit on any of these computers alone. The other obvious advantage of such a parallel implementation is the expected higher execution speed.

Concerning functional extensions, we are working on the explicit modeling of intersections—not present in the current model where intersections are modeled implicitly. The first track followed here is the modeling of traffic signals. Our model can be extended to handle red phases by simply reducing the outflow capacity of the corresponding link to zero. During the succeeding green phase, the original capacity is then restored. The second track is to incorporate general intersection dynamics based on direct interactions between cars. The idea here is to limit the total traffic flow through an intersection. As the maximum flow is limited, all road segments crossing at an intersection are competing. A large load in one direction will therefore reduce the maximal throughput in all other directions.

5.8 Summary

We have presented a traffic flow microsimulation using a queue-based and event-driven approach jointly. Compared to cellular automata, this process does cause a certain reduction in spatial resolution. However, that is acceptable compared to the very significant speedups gained by this approach. Compared to earlier queue-based approaches, our event-driven simulation saves time in areas of the road network where the traffic load is moderate to small. Overall, a speedup factor of 20 can be expected when compared to earlier queue-based approaches.

In addition to acceleration of the method, several other modifications were undertaken. The simulation now accurately handles backward-traveling gaps, limits the inflow capacities of all road segments in a meaningful way, and handles gridlock in a new way in the simulation by using a notion of guaranteed minimal capacities.

The final simulation was tested with a large-scale 24-hour scenario with seven million simulated person-days on a network with 28k links. The tests were undertaken on a single CPU workstation computer with a current processor and 4GiB of RAM, of which 3.2GiB were used. The simulation took about 53 minutes to finish, corresponding to an overall real-time ratio of 27.

Chapter 6

An Event-Driven Parallel Queue-Based Microsimulation for Large Scale Traffic

Authors

David Charypar
Institute for Transport Planning and Systems, ETH Zurich, Switzerland
Email: charypar@ivt.baug.ethz.ch

Kay W. Axhausen
Institute for Transport Planning and Systems, ETH Zurich, Switzerland
Email: axhausen@ivt.baug.ethz.ch

Kai Nagel
Institute for Land and Sea Transport Systems, TU Berlin, Germany
Email: nagel@vsp.tu-berlin.de

Published in

This paper was presented at a conference, see Charypar *et al.* (2007a).

Abstract

Traffic flow microsimulations are interesting for transport planning problems due to their high temporal and spatial resolution. Unfortunately, most of them involve high computational costs making them impractical for running large scale scenarios. We present a parallel microsimulation software that uses queue-based link dynamics together with event processing instead of time-steps. By introducing appropriate load balancing and by minimizing interfaces between processors to reduce communication needs as far as possible, our simulator requires only 87 seconds on 64 processors to simulate a 24 hours test scenario with 7 million simulated person days on a network with 28k links.

Notice to the Reader

The chapter at hand describes the parallelization of the traffic flow microsimulation the sequential version of which was presented in the previous chapter. Naturally, is contains some overlap with that chapter, namely where the underlying model needs to be explained. This is inevitable as otherwise the parallelization steps cannot be properly understood. The reader who has already read the previous chapter will profit most of Sections 6.6, 6.7, and 6.8 of this chapter.

6.1 Introduction

Traffic flow simulation is an important problem in transport planning, as it represents the final link between the description of travel demand and the emergence of flow densities, volumes, and travel speeds. In today's practice, traffic flow is most often simulated using aggregated models that are easy to use and well established in the community. However, compared to aggregated models, traffic flow microsimulation has certain clear advantages:

- Very high spatial and temporal resolution. Features like traffic jams and rush hours can be captured accurately.
- Traffic is represented in a natural way; vehicles traveling, roads, and intersections are simulated directly.
- Traffic flow microsimulation can be easily coupled with other microscopic approaches, e.g. agent-based demand modeling.
- Inverse analyses can be carried out to determine where certain cars are coming from and why they can be found at a specific road network location.

However, in many cases traffic flow microsimulations have prohibitively high computational burden especially for very large scale applications with more than one million person days simulated on high resolution networks. This is even true when using parallel machines to solve the computational task. As a result, microsimulations are most often only used for small applications and aggregated methods are used for large scale problems.

This work aims to present a new event-driven queue-based approach that is parallelizable and makes the traffic flow microsimulation of very large-scale problems feasible on affordable computer hardware. To boost simulation speed and size of the scenario the option exists to execute the software on large parallel machines with processors added as necessary. This also means that our code profits from multi-core processors which are becoming common. We show that the software performance scales nicely with the number of processors for systems with up to 64 processors, making it possible to run traffic flow microsimulations 200 to 500 times faster than time-step based queue-based approaches on single CPU computers.

6.2 Related Work

In this section, we present a selection of previous work related to our microsimulation. There are many different traffic flow simulation approaches. The physically based microsimulations (e.g. AIMSUN (Barceló *et al.*, 1998; AIMSUN, 2006), MITSSIM (Yang, 1997; MITSIM, 2006), VISSIM (VISSIM, 2006)) generally try to capture as many traffic flow phenomena as possible. They simulate car following and lane changing behavior and use a continuous representation of space and constant very small time-steps to simulate cars on the roads. AIMSUM (Barceló *et al.*, 1998) presented an implementation of a parallel traffic simulation using threads. In their shared memory approach, all variables are globally accessible by all processors. They simulated small scale scenarios on 8 CPUs and reached a speed-up of 3.5 compared to the single CPU solution.

A different microscopic, but less physical, simulation approach is represented by cellular automata (e.g Brilon and Wu, 1998), used for example in TRANSIMS (Nagel *et al.*, 1998; Nagel and Rickert, 2001; Rickert and Nagel, 2001; TRANSIMS, 2006). Here, cars move through cells that they can occupy. Although we have a coarser level of detail in cellular automata, features like densities and travel speeds still emerge from the cars' simple direct interactions and are not computed at an aggregated level. The parallel version of TRANSIMS (Nagel and Rickert, 2001; Rickert and Nagel, 2001) used message passing between processors, running midsized scenarios on 32 CPUs. Although the run times were about 10 times faster than real time, ethernet latency problems and speed handicaps due to the use of cellular automata were reported. Using the same parallel concepts as in TRANSIMS, the queue-based model presented in (Cetin, 2005; Cetin *et al.*, 2003) achieved a speed-up of 32 using 64 CPUs.

In mesoscopic models (e.g. METROPOLIS (Marchal, 2001; de Palma and Marchal, 2002), DynaMIT (Ben-Akiva *et al.*, 1998; DynaMIT, 2006), DYNASMART (Chang *et al.*, 1994; DYNASMART, 2006), DYNEMO (Schwerdtfeger, 1984; Nökel and Schmidt, 2002), ORIENT/RV (Axhausen, 1988)), vehicles are still represented microscopically, but in contrast to the microscopic approaches described above, travel times and speeds are calculated using aggregates. By using a parallel implementation based on threads, METROPOLIS managed to simulate large scenarios efficiently. DynaMIT uses functional decomposition (task parallelization) as parallelization concept. Different modules are run in parallel, but the traffic simulation is executed on a single CPU only. The parallel implementation of DYNEMO (Nökel and Schmidt, 2002) used message passing on 19 CPUs for simulating small scenarios. Larger numbers of CPUs were reported to be inefficient.

The highest level of abstraction can be found in macroscopic models that compute all traffic quantities on an aggregated level. One example of a traffic simulation model as a one-dimensional incompressible fluid is NETCELL (Cayford *et al.*, 1997).

The work presented here builds on the approach of MATSim-T (Cetin, 2005; MATSim-T, 2006). While using simplified, queue-based dynamics to be computationally efficient, MATSIM-T still estimates all quantities microscopically and thus should be located somewhere between mesoscopic approaches and cellular automata. The same holds for the approach presented in Mahut (2000) where the computational effort on links is minimized by only calculating the entry and exit times of vehicles.

Time-step based approaches have been more popular in the past than event-driven approaches. It is not clear why, but one reason might be the more straightforward implementation of time-step based simulations.

In this work, we extend an event-driven approach presented in Charypar *et al.* (2007c) using parallelization for the simulation of larger scale scenarios.

6.3 Classification of Traffic Flow Microsimulation Methods

6.3.1 Cellular Automata

Cellular automata are used in traffic flow microsimulation to accurately model the behavior of cars traveling on a road network (see e.g. Chowdhury *et al.*, 2000). The basic idea is to discretize space in cells of equal size, each of which can be occupied by, at most, one vehicle. The cars drive through these cells by choosing their speed according to the space available in front of them. Cellular automata have the advantage of simplicity while still retaining a fair amount of spatial resolution, which - among other things - has the advantage that link capacities are generated from the properties of the links and the behavior of the drivers.

The major drawback of this method is its computational cost as every agent is simulated in every time step (usually one second). This becomes impractical if we are interested in large scale simulations.

6.3.2 Queue-Based Simulations

If the computational speed of a model is not sufficient for a certain kind of application, one solution is to switch to a simpler one. We follow this approach in MATSIM-T, see for example Cetin (2005); MATSim-T (2006) where we use a queue-based approach to achieve higher performance. Here, the basic assumption is that the main features of (at least) urban traffic can be modeled by looking at the intersections alone, resulting in a model where:

- Links are simulated as they "process" cars traveling through the network.

- A queue stores cars traveling on each link together with their respective entry times.

- Each link's capacity and space available for cars are parameters of the model.

- To move cars through the network, consecutive links collaborate in each time-step monitoring various constraints (capacity, free speed travel time, intersection precedence, and space available at the next link).

Queue-based models are faster than cellular automata mainly because the number of simulated units is smaller (links vs. cars), thus we gain roughly a factor of 10 to 100 depending on the network resolution.

6.3.3 Event-Driven Queue-Based Simulations

To boost simulation performance further, we seek to eliminate inefficiencies in the queue-based approach. One such inefficiency can be observed when low density traffic regions of the simulation are investigated: while each link is processed in every time-step, most of the time no cars are really moved.

In order to achieve higher performance, we chose to substitute constant time-steps for the direct treatment of actions occurring on the road network. Each such action is reflected by an event. Every time a car enters or leaves a link, a corresponding event is processed, meaning that the event processing rate is proportional to the traffic flow at any time. The two main advantages of the event-driven approach are:

- Processing time is assigned according to the traffic flow during any time of the day. As a result, most computational time is spent where traffic flow is maximal and almost no time is spent where the traffic network is empty.

- Prediction of processing time for a given travel demand is easy as it is proportional to the overall traffic volume simulated.

The event processing mechanism introduces an overhead compared to time-step based approaches that might be a concern in situations with large traffic flow. However, so far we have not come across a situation where the time-step based approach outperforms the event-driven simulation even when simulating only rush hours. For 24 hour scenarios, one can usually expect an overall speedup factor of 10 or more over the time-step-based implementation.

	Spatial discretization	Temporal discretization
Cellular Automata	equidistant	constant time-step
Queue-based simulation	road segments	constant time-step
Event-driven, queue-based simulation	road segments	adaptive, event-driven

Table 6.1: Discretization schema of different microsimulation approaches

In METROPOLIS (see de Palma and Marchal, 2002), a similar, event-based approach is used. The important difference there is that travel times and speeds are estimated using aggregated values, while in our implementation, no aggregates are used.

A comparison of the simulation approaches' spatial and temporal discretization schema is shown in Table 6.1.

6.4 Modifications to Queue Dynamics

In the process of implementing a parallel event-driven approach we have also added several features not present in our former implementations of queue-based microsimulations:

- *Gaps, gap travel speeds*: The new implementation uses a concept of gaps that travel backwards through road segments at a constant speed. They are used to produce a delay between the event of a car leaving the road segment at the downstream end and the time the formed gap becomes visible at the upstream end. The benefit of this approach is mainly that it improves the way congestion dissolves.

- *Capacity constraints at inlet*: In addition to the outflow capacity, the inflow capacity of each link is constrained in the new implementation. This improves the behavior at converging Y-type connections where in former implementations breakdowns were predicted too far downstream.

- *Handling of gridlock*: All microsimulations have to cope with the issue of gridlock that happens if agents block each other at intersections. We solve this problem by guaranteeing a certain minimum inlet capacity for each road segment. This capacity is available even if the road segment is already full allowing it to temporarily overfill. By permitting additional agents to enter a link they are removed from the problematic intersection which in turn averts gridlock.

6.5 Software Design: Technical Description

In the following section, we present the different parts of our event-driven queue-based traffic flow microsimulation and we explain how these parts work together to solve the simulation problem at hand. The subsequent parallel implementation of our model is covered in the next section.

6.5.1 Description of Travel Demand

For the input travel demand to our traffic flow microsimulation we employ the formalism used in Balmer *et al.* (2006b):

- We use a population of agents, each having a complete 24-hour activity plan.
- An activity plan consists of an activity pattern describing the type and order of activities executed, location choice information describing where the activity in question should be executed, timing information defining the time of day when the activities are taking place, and routing information describing the sequence of road segments taken to travel between subsequent activities.

6.5.2 The Traffic Flow Model

The basic concept of our traffic flow model is that we only simulate transitions of agents from one road segment to the next. During the time spent on a specific road segment, very little information about the position of the agent is available: only the position in the queue is stored and the earliest time that the car could leave the link according to free speed limitations. When the agent arrives at the end of a road segment, an event occurs and the agent leaves the road segment and enters the subsequent one according to the route stored in his complete activity plan. However, this can only happen if there is sufficient space left on the downstream link to take the oncoming agent. Otherwise, the agent has to wait until sufficient space is available.

Travel times derive from the behavior of the agents on the link: if a link is empty, the travel time equals the time needed to drive down the link at free speed. The travel time can lengthen if there are other agents traveling in front of the agent. In this case, the agent cannot leave the link before all agents in front of it have left the link. The reason for such a delay is usually high demand coupled with limited outflow due to limited outflow capacities or a blocked road network downstream. That means, as expected, large traffic volumes coupled with low link capacities will lead to long travel times.

6.5.3 Simulation Output

The primary output of our traffic flow microsimulation is a log file that lists each event during the course of the simulation run. The complete description of an event consists of the absolute time of day when the event happened, an agent-ID and a link-ID specifying which agent was involved and where the event happened, respectively, and the type of event e.g. entry or exit. Using simple post processing, this events file can be converted into count data, hourly volumes on links, person diaries, or data similar to GPS floating-car data.

6.5.4 Actors During the Course of the Simulation

In our simulation software, three basic elements combine to compute the traffic flow through the road network (see Figure 6.1). These three elements are:

- The road segment that holds the cars during their travel,
- The agent that: 1. represents the object moving through the road network and 2. stores all information about the travel demand.

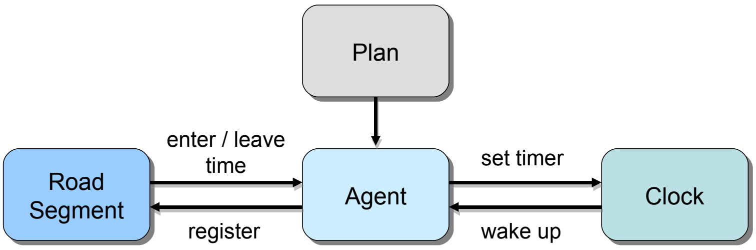


Figure 6.1: Illustration of interaction processes: the boxes represent actors in our microsimulation, the arrows stand for flow of information and message passing.

- The clock that handles registered timers and alerts the agents when an action must be undertaken.

6.5.5 The Traffic Flow Protocol

The actors mentioned above all interact and communicate according to a certain protocol explained and illustrated by the example in Figure 6.2, where one agent wants to travel along a certain sequence of road segments. The protocol consists of the following steps:

1. The agent informs the road segment that it wants to enter. The road segment stores this request chronologically together with all other requests of this kind.

2. As soon as it is clear when a gap will be available for the agent, the road segment sends this entry time to the agent.

3. The agent registers a timer for this time with the clock.

4. As soon as the timer expires, the clock sends a wake-up call to the agent and informs it about the time.

5. The agent then enters the road segment. With this step, the first part of the protocol finishes, and the agent continues to travel down the road. Note that this does not require any actions to be taken by the road segment, the agent, or the clock.

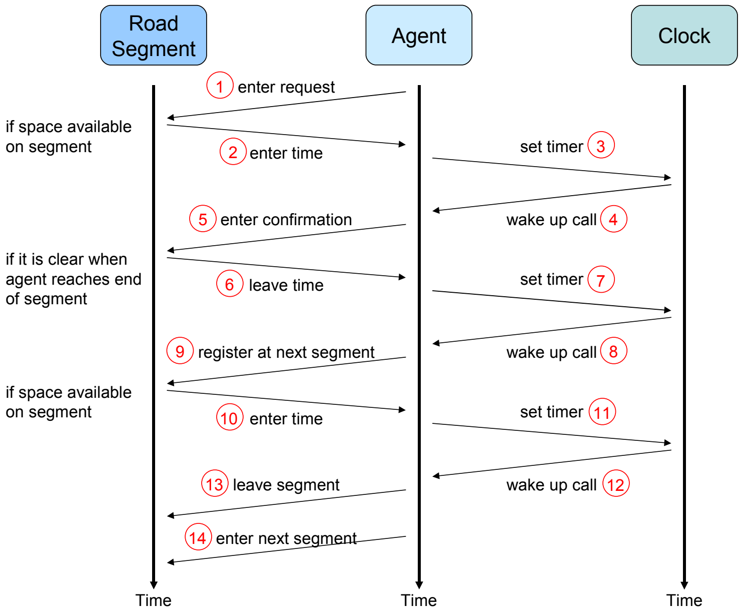


Figure 6.2: The traffic flow protocol: the vertical lines indicate how time runs during the course of the simulation; the arrows between these lines represent messages sent between the actors of the simulation; the numbers in circles denote the corresponding protocol step of the message. See text for more information.

6. When the agent moves up to the first position in the road queue, the road segment computes the time the agent is going to arrive at the end of the road segment and sends this information to the agent.

7. Similar to step 3, the agent registers a timer with the clock.

8. After the timer expires, the clock calls the agent.

9. Now the agent is at the front of the road segment. It then checks with its activity plan for the next road segment to travel and registers there.

10. When it is clear at what time there will be a gap available for the agent to enter the next link, it receives a message with the predicted entry time.

11. The agent again registers a timer for this time.

12. When the timer expires, the clock wakes the agent.
13. The agent informs the last road segment that it is leaving.
14. It also informs the next road segment that it enters. From here on, steps 6 to 14 repeat until the last road segment of the trip is reached; then, step 14 is left out and the protocol stops.

6.6 Parallelization

Using queue-based link dynamics and an event-driven approach running large scale scenarios (with roughly one million person days simulated on a network with roughly 10k links) becomes feasible on single CPU desktop computers (Charypar *et al.*, 2007c). However, when going to even larger scenarios (roughly 10 million agents on high resolution networks with 100k links and more) the computational burden and memory consumption again become an issue. This is especially true if we want to iterate 20 times or more during the search for a user equilibrium. To be able to do this for very large scale scenarios it is necessary to speed up the traffic flow simulation even more as we have less than 30 minutes for one simulation run if we want to finish overnight. To achieve this further reduction in processing time and also to cope with the memory demand we use parallelization to spread the simulation across multiple processors of a parallel computer.

6.6.1 Domain Decomposition

In order to distribute the computation across multiple processors we decompose the simulation domain (i.e. the network). The basic idea is to subdivide the network into parts and assign all nodes residing in one part to the same processor. Road segments that connect nodes assigned to the same processor are simulated entirely on that processor while road segments connecting nodes on different processors are simulated on those two processors jointly involving periodical communication.

To achieve reasonable parallel speedup it is important to have equal workload on all processors and to minimize the interfaces between individual parts of the domain to reduce the communication needs to a

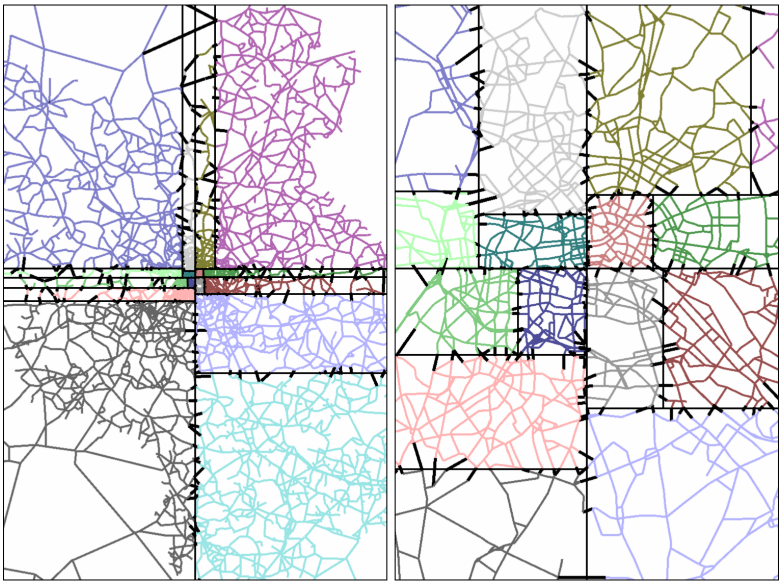


Figure 6.3: Domain decomposition of the road network of the federal states of Germany, Berlin and Brandenburg: On the left you can see a complete view while on the right a close up of the center section is shown. Each colored area corresponds to the same traffic load and represents the assignment to a specific processor for the parallel simulation using 16 processors. The thin black lines represent the domain boundaries. Thick black lines indicate links that cross subdomain boundaries and involve communication.

minimum. To achieve these two goals, the domain is partitioned using orthogonal recursive bisection, selecting the splitting plane such that the daily traffic volumes in both parts are equal. For this process, we use load data available from a previous iteration (Figure 6.3). This decomposition of the simulation domain is a simple, practical and efficient algorithm that can be implemented quickly. The domain decomposition also handles how the memory demand of the whole simulation is distributed across processors: at any time each processor holds solely agents currently residing on a link in care of that processor. This distributed handling of information allows to make use of the memory resources available on all processors together.

6.6.2 Communication

The only parts of our parallel program that involve communication between processors are road segments crossing the boundaries between subdomains (indicated as black links in Figure 6.3). In our program, such a road segment is split into two parts - the road start and the road end - which are then simulated separately on the two processors involved. From time to time synchronization messages are sent between the two parts of the same road segment. This is done to exchange information about agents that entered or left the road segment since the last synchronization. If agents travel across the simulation boundary between two processors these agents including their daily activity plan are packed into the synchronization message and thereby transferred to a different processor. This means that agents starting on one processor may go to any other processor during the course of the simulation.

The synchronization messages between the two parts representing a road segment have to be issued often enough such that the individual parts cannot become invalid. On the other hand, we want to communicate only as often as necessary in order to keep communication overhead as low as possible. Fortunately, the synchronization interval for each road segment (the time between two subsequent synchronization messages on one road segment) can be chosen independently of any other road segment. It depends solely on the speed of information propagation across this specific road segment. As the only information transported across road segments are agents traveling and the gaps that travel in the opposite direction, it is the free speed travel time of agents and gaps that give us the desired synchronization interval. [1] For a given road segment this value is given by the length of the road segment and the larger of the free speed travel time and the gap travel time.

In addition to the synchronization interval, we have to define at what specific times synchronization messages are sent. It is important that corresponding parts of the same road segment send and receive messages at the same time. Our solution to this problem is to start all processors at the same simulation time (the beginning of the period to be simulated,

[1] This means that the parallelization is greatly facilitated by the presented explicit handling of gaps. One could argue that it is this property which gives this type of simulation a better parallel efficiency than other approaches.

e.g. midnight) and communicate each time a synchronization interval has passed.

6.6.3 Technical Description

Our software is implemented as an explicitly parallel program. We have used the Message Passing Interface (MPI) which is a quite flexible solution that makes it possible to run our code on all kinds of parallel computers including shared memory architectures and computer clusters. However, the performance is limited by the communication possibilities between processors and therefore it is to be expected that our software has larger potential for parallelization on shared memory computers than on computer clusters connected using cheap networking components.

6.7 Results

6.7.1 Test Scenario

The test scenario we use to demonstrate the properties of our simulation software consists of the following parts:

- The road network of the federal states of Germany Berlin and Brandenburg. It consists of 11.6k nodes and 27.7k links.
- The synthetic population of the area consists of 7.05M people. Each person has a complete daily activity plan with multiple activities and trips. That means we are simulating 7.05M person days.

The average number of trips per agent in our demand is 2.02 and the average length of a trip is 17.5 links. This leaves us with an overall daily demand of 249M road segments to be traveled.

6.7.2 Computer System for Performance Analysis

All our tests were run on a shared memory parallel computer equipped with 64 dual-core Intel Itanium 2 processors with 1.6GHz and a total of 256GiB of RAM.

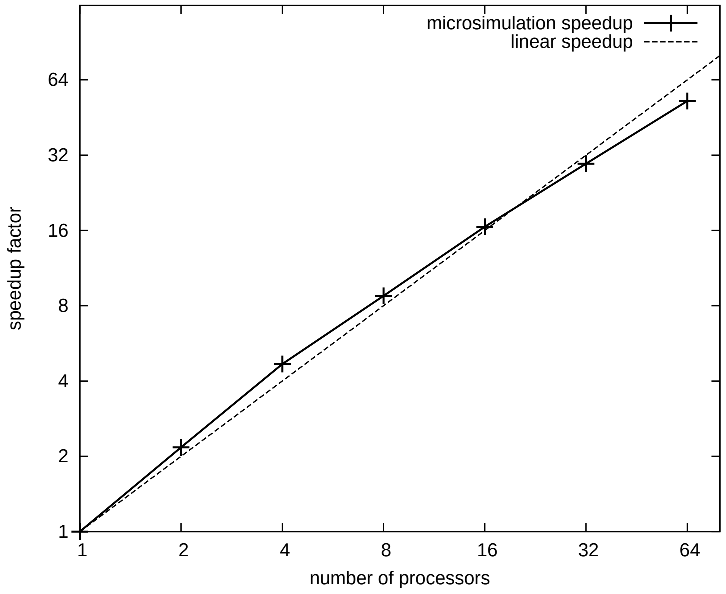


Figure 6.4: The software speed scales nicely with the number of processors used. Note the superlinear speedup for up to 16 processors probably due to better cache efficiency. For 64 processors, the speedup reaches roughly 53.

6.7.3 Test Results

We ran the test scenario on the machine mentioned above. An average run using one CPU core took roughly 15 minutes to read in the travel demand, 25 minutes to produce the output file (events file) and 77 minutes to actually compute the traffic flow over the day. The following performance data refers to the computation time for the traffic flow, disregarding all input and output operations. Figure 6.4 shows how the performance of our simulation scales with the number of processor cores used. It can be seen that the system scales nicely using up to 64 processor cores where the performance is roughly 53 times the single core performance. The best parallel efficiency can be observed with 4 processor cores and with up to 16 cores the simulation runs with superlinear speedup. With 64 processor cores it is possible to run our test scenario in 87 seconds. Note that the speedup still increases around 64 processor cores and it might be possible to go beyond that point on even larger machines.

6.8 Discussion and Outlook

We have shown that our method is able to simulate large scale scenarios as our test scenario of Berlin and Brandenburg at satisfactory speeds. However, in our current project, we aim at simulating a very large scale scenario where all people in Switzerland (roughly 7 million agents) are simulated on a high resolution network with a size in the order of 500k links. Using the property of our simulation that the processing time only depends on the total traffic volume, the processing time can be estimated once this volume is known approximately. We estimate that the scenario describing all of Switzerland will be about a factor of 10 larger than the test scenario used in this paper. This would mean that it is feasible to compute the high resolution traffic flow for all of Switzerland within roughly 15 minutes on 64 processors. However, handling the tremendous amount of data related to these scenario sizes remains a major challenge. Currently, when running the simulation using 64 processors, more time is spent for input and output routines than for actually simulating traffic flow. Further work has to be done on this to reduce I/O needs of the simulation and to improve parallel I/O bandwidth.

Concerning functional extensions, we are working on the explicit modeling of intersections. This is not present in the current model where they are modeled implicitly. The first track followed here is the modeling of traffic signals. Our model can be extended to handle red phases by temporary reducing the outflow capacity of the corresponding link to zero and restoring the original capacity with the next green phase. The second track is to incorporate general intersection dynamics based on direct interactions between cars. The idea here is to limit the total traffic flow through an intersection. As the maximum flow is limited, all road segments crossing at an intersection are competing. A large load in one direction will therefore reduce the maximal throughput in all other directions. We believe that these extensions will lead to more realistic flow patterns.

6.9 Summary

We have presented a traffic flow microsimulation using a queue-based and event-driven approach jointly. We chose our approach to achieve a significant speedup compared to e.g. cellular automata, accepting a certain loss in spatial resolution to be able to solve large scale traffic simulation problems. Compared to earlier queue-based approaches, our event-driven simulation saves time in areas of the road network where the traffic load is moderate to small leading to a speedup compared to the time-step-based approach of more than 10 for 24 hour scenarios.

In addition to accelerating our method, several other modifications were undertaken. The simulation now handles backward-traveling gaps, limits the inflow capacities of all road segments in a meaningful way, and handles gridlock by using a notion of guaranteed minimal capacities.

By using a suitable domain decomposition that balances the load on the processors and minimizes communication interfaces, we succeeded in parallelizing our software which makes it possible to use parallel computers to boost the performance by at least another factor of 50. The test scenario used was a large-scale 24-hours scenario with seven million simulated person days on a network with 28k links.

Chapter 7

Fundamental Diagram

7.1 Introduction

In the previous two chapters we have presented a queue-based microsimulation with gaps and shown that our implementation is efficient enough to simulate the traffic flow in a large region with around seven million travelers. The main reason for the good performance is the event-driven approach which saves a lot of time when simulating off-peak hours and areas with strong congestion.

Compared to time-step-based implementations, there should be no negative effects by using events. Actually, the temporal resolution of the method has increased from seconds to a continuous representation of time, giving smoother results than before. We also believe that introducing backwards traveling gaps should in fact enhance the properties of the traffic flow simulation. Still, a rigorous investigation of these properties is missing. Such characteristics are important to know, particularly to estimate for which problems and how our traffic flow microsimulation can be used.

In this chapter, we systematically analyze the flow characteristics of the microsimulation at hand. Then, we use these results to make recommendations on how the microsimulation should be used. Knowing this is especially important when simulating freeway traffic. A typical real-world flow-density relationship is shown for instance in (Nagel *et al.*, 1998)

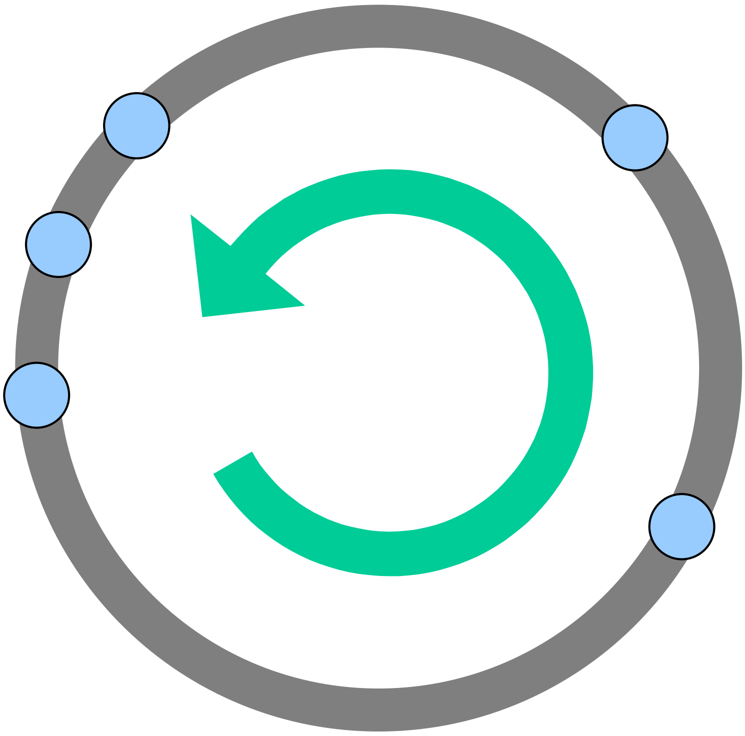


Figure 7.1: A virtual test setup with agents (blue) living and traveling on a ring test network

7.2 Average Flow Characteristics

The basic idea to measure the flow characteristics of the presented microsimulation is to use a virtual experiment with test agents on a test network (see Figure 7.1), and to analyze the resulting data (events) in terms of flow speed, density, and volume.

7.2.1 Test Network

We create parametrized test ring networks to perform resolution tests on the same topology. The ring road has a length equivalent to 1024 parked cars. Assuming an average space need of $7.5\mathrm{m}$ per car, this results in a circumference of $7.5\mathrm{m} \cdot 1024 = 7680\mathrm{m}$.

To produce test networks of variable resolution that could be easily compared, the ring was subsequently dissected into a power of two num-

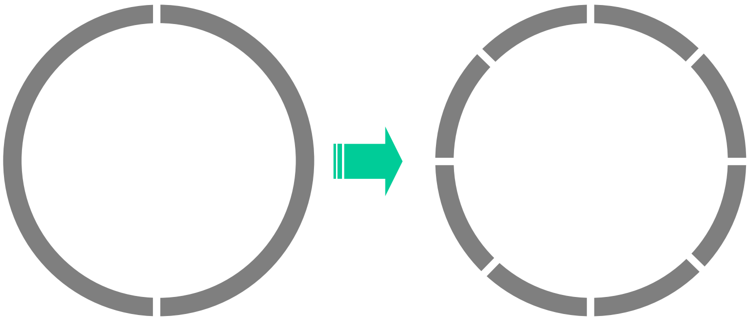


Figure 7.2: The test networks are created at resolutions from 2 links up to 256 links.

ber of road segments. (See Figure 7.2) The number of segments ranged from 2 to 256.

7.2.2 Test Agents

In our test scenario, there is a number of 1023 agents that all share a very similar activity plan. All agents live at a random position on the road network. At a random point in time, they start to travel around the road network and continue for 100 times. At the end of their journey, they arrive at the same place as they started before.

From the networks point of view, this behavior results first in a continuously increasing number of agents traveling, until all are on the road. During this increase, we are able to measure flow properties at increasing densities. When all agents are on the road, we can observe maximally congested road conditions. At the end, when agents start arriving, the density decreases, giving us the chance to observe flow at reducing densities.

7.2.3 Measured Values

In general, we are interested in the relationship between three different quantities: flow density, flow speed, and flow rate of the observed traffic.

In the first part of this chapter, we are interested in quantities averaged over a certain amount of time. The values observed directly are the

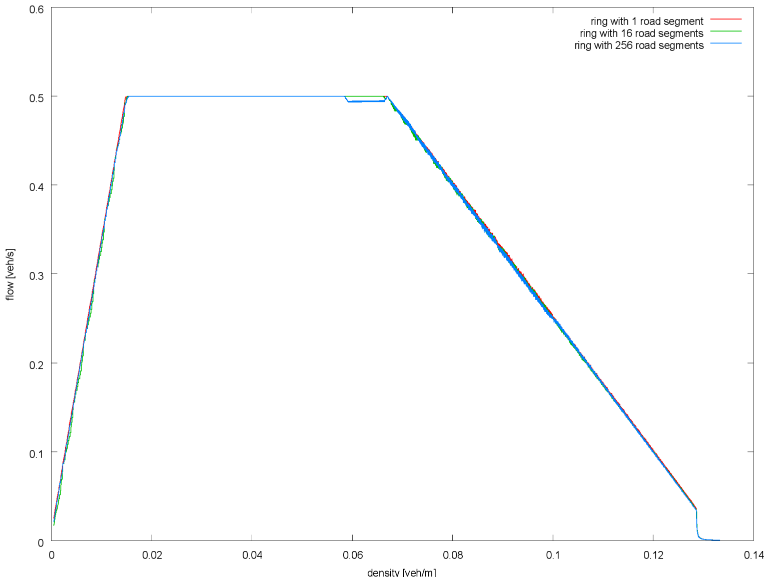


Figure 7.3: In our queue-based microsimulation model with gaps the average density to flow ratio show a clear trapezoidal relation. This property is independent of network resolution.

time of circulation and the average flow density (the number of cars on the network) during this time. The average travel speed can then be computed from the network circumference and the circulation time, and the average flow rate can be computed from average density and circulation time.

7.2.4 Results and Discussion

In Figure 7.3, we show the measured mutual dependence between traffic density, flow, and speed. The graph has a trapezoidal shape, the left, upper, and right edges of which can be clearly associated with independent parameters of the queue-based model with gaps: the slope of the left edge is controlled by the free speed defined on a road segment, the height of the top flat part corresponds to the flow capacity on the link, and the cut-off point together with the slope of the right edge are a result of the maximum density (fully congested density) and the backward traveling gap speed.

The plots were generated for all test network resolutions showing no apparent differences in the flow characteristics. It seems that network resolution does not produce an immediate effect on the average flow properties. It is interesting that all observed states on the network lie on the border of the graph. No state of medium densities together with medium flow was observed in all our tests. It seems that our model is not only able to produce the limit values, but it also *will* produce exactly these limit values if used on a ring topology.

7.3 Instantaneous Flow Characteristics

At this point, one might ask for more variation in the produced flow patterns. Particularly, it is not clear what the reason is that all observed states lie on the envelope of the flow-density diagram and no states inside the trapezoid can be observed. Essentially, two entirely different explanations are possible: 1. There are no fluctuations at all. The state of the road segment always moves along the envelope of the graph. 2. There *are* fluctuations producing states inside the envelope, realizing non extremal values of the state variables, but subsequent smoothing (as it was done in the acquisition of the data) then eliminates the variation originally available.

7.3.1 Modifications to Measure Instantaneous Values

We modify the test setup to enable measurements of the instantaneous state of the road segment. This is done by introducing a small sensor link just long enough to hold one car ($= 7.5\mathrm{m}$) (see Figure 7.4). We register the time each time a car enters or leaves the sensor link. The instantaneous flow can be computed from the time between two subsequent cars leaving the sensor link. The travel speed can be measured through corresponding entry and exit events on the sensor link. The density must be measured indirectly since the sensor link can only either contain or not contain a car. Therefore, we plot the traffic density ρ as $\rho = q/v$, where q is the traffic flow and v is the traffic speed.

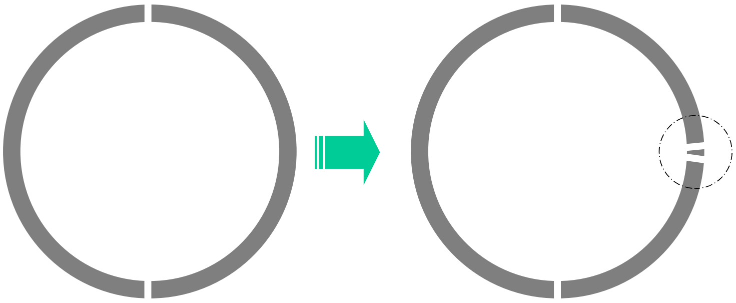


Figure 7.4: To measure instantaneous flow characteristics of our model a short sensor link is added to the ring test network.

For this investigation we can leave the demand (the agents plans) the same. Only the space on the ring is slightly increased from 1024 to 1025 cars by the additional sensor link.

7.3.2 Changes in Results and Discussion

Figure 7.5 shows a very similar picture compared to the original Figure 7.3. The edges are still very prominent, again with parameter values clearly visible. Interestingly, almost all observations show either very high or very low density. Medium density observations are almost absent. Furthermore, almost all data points still reside on the envelope meaning that the model moves away from extremal values only very rarely.

7.4 Stochastic Flow characteristics

According to the findings of the last section, in the ring test scenario the queue-based model basically reproduces the envelope of what could be termed a stylized fundamental diagram. But one question remains: are the “inner” states not possible or is the test setup design avoiding these traffic states?

To demonstrate the general ability of the model to produce any combination of state variables we finish the investigation by using a stochastic demand setup.

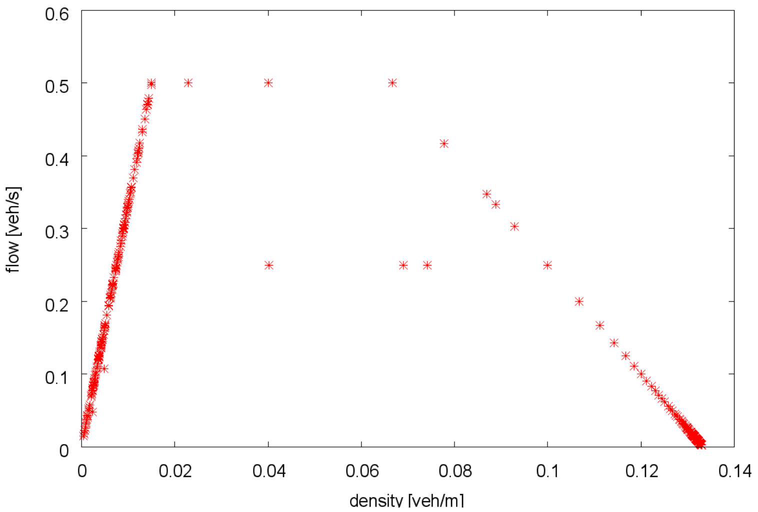


Figure 7.5: The measurement of instantaneous values reveals the same trapezoidal relation as in the averaged case with a concentration of data points at the left and right borders.

7.4.1 Stochastic Demand and Supply

The basic idea is to replace the test road ring by a single short ($7.5\mathrm{m}$) test road segment. The agents including their travel plans are replaced by two random processes: 1. Cars are generated randomly at the upstream end of the road segment such that the gap between subsequent desired entry times is a random distribution in the range $[T_{\text{min,car}}, \infty)$, where $T_{\text{min,car}}$ is the minimum gap according to inflow capacity restrictions. 2. The gaps moving upstream from the downstream end of the link are generated randomly, such that the time between two subsequent gaps is in the range $[T_{\text{min,gap}}, \infty)$, where $T_{\text{min,gap}}$ is the minimum gap according to outflow capacity restrictions. The final measurements are done by using subsequent exit events (when cars leave the measuring segment) to compute traffic flow q and by using corresponding entry and exit events to estimate speed v. As before, density ρ is inferred from these two values using $\rho = q/v$.

7.4.2 Observations

In Figure 7.6 we show the results with stochastic demand. The plot

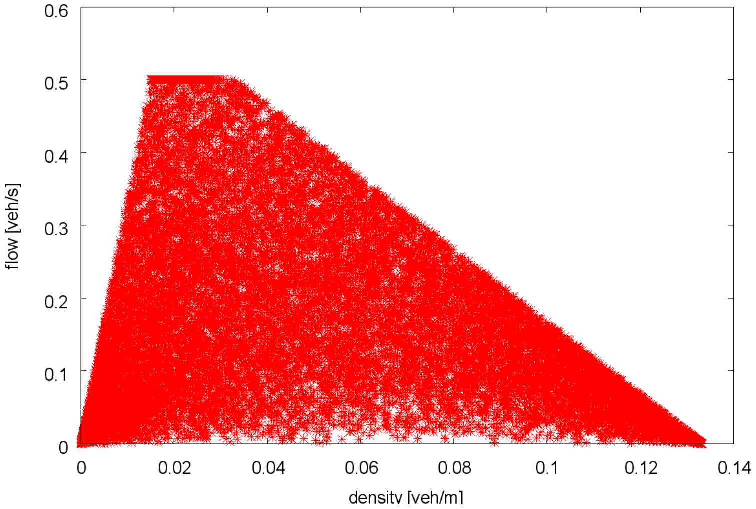


Figure 7.6: Using stochastic demand (agents arriving upstream, gaps arriving downstream) the "inner states" can be reached using our queue-based traffic model.

shows the same trapezoidal envelope with the same interpretations of free speed, capacity and gap travel speed as in the previous two cases. In contrast to the deterministic demand cases, here, the inner states are realized regularly. It seems that the random process producing the demand is sufficient to produce basically any state inside the envelope.

7.5 Conclusions on Flow Characteristics

Based on the three tests we have carried out in this chapter we draw the following conclusions:

- The interrelation of traffic flow and density on the simulated road segments shows a trapezoidal envelope. This envelope is defined by four independent parameters of the queue-based model: free speed, capacity, gap travel speed, and maximum density.

- The network resolution of a test ring scenario has no effect on the resulting averaged density and flow patterns. We conclude that it is sufficent to simulate freeway traffic by using relatively long road segments. Therefore, the limiting factor in terms of spatial resolution should be merely the desired resolution of the activity locations.

- If the demand is deterministic the instantaneous state of the road segments almost never reaches points inside the envelope (at least in the ring test scenario).
- If there is sufficient fluctuation in the demand and in the backward traveling gaps (which are also induced by the demand) essentially any traffic condition can be reached (i.e. also inner states are possible).

Chapter 8

Green Time Fractions

The queue-based microsimulation approach described in this thesis is a link-based aproach. That is, the model's dynamics originates from the links and not from the nodes. Also, an agent or car situated on a link can never know where on that link it is exactly. The spatial resolution of link-based approaches is one link and therefore network resolution dependent. Agents start and end their trips on the links, and the whole time during a trip each agent finds himself on one or another link. Where links are connected to other links, nodes form. In our microsimulation, these nodes are represented only implicitly, without any behavior directly related to them. Rather, they are modeled distributedly by all the links connected at a certain point. The capacity constraints of each node are maintained by observing the relevant flow capacities of incoming and outgoing links and by sharing the available capacity according to demand.

The simulation of 24 hours scenarios on urban networks with many signaled intersections often involves a special problem: The observable link flow capacities on crossing roads do not always keep a fixed ratio as the signal control parameters may be time of day dependent. This type of intersections cannot be handled correctly throughout the day, not even theoretically. To alleviate this problem, we introduce green time fractions that modulate the links outflow capacity. These green time fractions can be varying over time, enabling the simulation of different traffic control schemes over the day.

8.1 Model

The general idea is to introduce on each link an explicit representation of the outgoing capacity reduction based on the fraction of time the associated traffic light is switched to green. We simply assume the average capacity realized on such a link is proportional to the relative green time. For example on a link with its traffic light switched to green during 30% of the time on average, we expect to observe an average realized capacity of 30% of the free flow capacity. Admittedly, there probably is no linear relation between green time fractions and realized capacity. Still we choose this model to make the simulation easy to understand. After all, we can switch to an interpretation of the data where we replace relative green time by relative *useable* green time, rendering our linear model correct.

In our model, the green time fractions per link are not constant but are merely modulated, each individually, by a piecewise linear function of time. Other models would have been possible (including piecewise constant modulation functions), but we think it is a beneficial property to have a continuously changing capacity on the links to avoid undesired shocks in the system. After all, if desired, very quick changes in capacity can still be realized.

The question might arise why we model averaged green time fractions instead of individual green phases. There are multiple reasons to do so:

- The necessary data is difficult to obtain which is especially problematic since in our work we are addressing very large scenarios. For these scenarios, green phase data of hundreds or even thousands of traffic lights would need to be obtained.

- This data cannot be easily made up since not only the green phase durations have to be found but also the delays between neighboring intersections. Small errors in these numbers may produce totally different realized flows on crossing links. Therefore, reproducing a certain flow rate using direct modeling of green phases is very difficult.

- For similar reasons, it is very difficult to test new policies using individual green phases compared to the relative ease of adapting the average green time fractions on a road during some time of day.

Note that if, for some reason, there is the need to simulate individual green phases on certain intersections, our model is flexible enough to mimic such a behavior by changing the green times fraction periodically. For example, a link's green time fraction can be set to alternate from 100% green for 73 seconds to 0% green (i.e. red) for 128 second by defining a corresponding modulation function in the input data.

8.2 Data Format

The following exchange format was defined to specify the green time fraction modulation function for each link involving a signaled downstream end. The program expects a separate file (a green time fractions file) the name of which must be specified in the configuration file of the traffic flow microsimulation. The XML-based format consists of a surrounding master tag `greentimefractions` containing a `linkgtfs`-tag for each road segment involving signaling. The periodicity of the modulation function is also specified within this tag. For each link a variable length list of "data points" is specified using the `gtf`-tag. Each data point consists of a time stamp and an associated effective green time fraction. The sofware automatically wraps around at the periodic boundaries. In Figure 8.1 you can see example green time fractions for link 123. A plot of the same data can be seen in Figure 8.2.

8.3 Technical Description

Internally, for each of the road segements holding green time fractions data an instance of a special modulator object is created. Apart from holding the data, this object is also responsible for computing the car waiting times on the associated road segment.

The data is stored in a sorted data structure allowing quick access to any data point needed during computations. Similar to the data format, each data point consists of a time stamp and a green time fraction.

```
<greentimefractions desc="Green times file">
     <linkgtfs id="123" time_period="24:00:00">
          <gtf time="05:30:00" val="0.4"/>
          <gtf time="07:00:00" val="0.6"/>
          <gtf time="09:00:00" val="0.6"/>
          <gtf time="10:00:00" val="0.4"/>
          <gtf time="16:00:00" val="0.4"/>
          <gtf time="17:00:00" val="0.3"/>
          <gtf time="19:30:00" val="0.3"/>
          <gtf time="21:00:00" val="0.4"/>
     </linkgtfs>
</greentimefractions>
```

Figure 8.1: An example green time fractions file

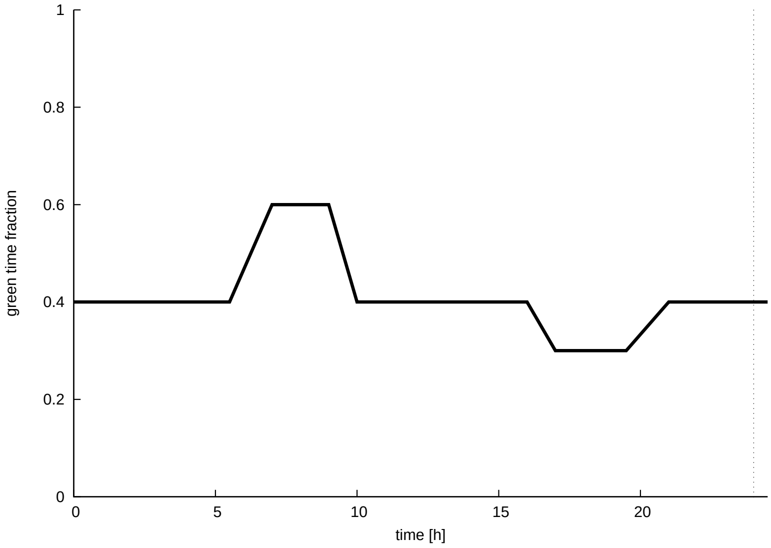


Figure 8.2: Plot of example green time fractions for link 123

When a car wants to leave a signaled road segment, it requests the next available time slot. This value depends on the time when the last car left the link and on the variable capacity of the road segment. Technically, the car has to wait until the integral of the time dependent capacity reaches $1[\mathrm{car}]$. This integral of the piecewise linear capacity function $q(t)$ is the piecewise quadratic volume function $V(t)$. If t_l is the time when the last car left the road segment, the requested next leaving time

t_n can be computed by solving the eqation

$$V(t_n) - V(t_l) = 1 \tag{8.1}$$

for t_n. This involves inverting the volume function which is not trivial, since it is a piecewise quadratic function, and it is not clear which piece will contain the t_n to be computed. Our implementation solves this problem by starting at t_l and integrating each piece of $q(t)$ sequentially until the integral exceeds the required $1[\mathrm{car}]$. At this point, it is clear that the last covered piece contains the t_n in question and this can be easily found by solving a quadratic equation.

8.4 Possible Use of the Method

If inclusion of new links into existing road networks is considered, for instance the creation of a city by-pass, accompanying measures are often investigated. Changing the timing and prioritization of traffic lights is often one of these measures tested and the presented extension to our traffic flow model provides a way of rendering such ideas in the simulation.

Chapter 9

Results

This chapter gives an overview of the results of the presented work from a more global point of view; at the same time, it tries to point out the relevance of the different methods for agent based traffic models used within the microsimulation groups at ETH Zurich and TU Berlin, as well as for agent based models in general. For detailed results on the specific topics, please refer to the corresponding chapter's result section.

9.1 Activity Planning

Activity based models consist of many parts (see Section 1.3) which we refer to as modules. Several of these modules have been developed or worked on during this thesis.

One of the most important parts of our agent based model is the activity planning process, and a major part of this thesis is devoted to it. First, in Chapter 2, activity planning of agents, including activity selection, location choice, and time allocation, is defined as an optimization problem of a utility assigned to such daily activity plans. For this, a simplistic yet powerful integrated utility function is designed, and set as a base of the activity planning process. To solve the complex, nonlinear, mixed discrete and continuous optimization problem, a custom-made genetic algorithm is designed, and it is shown to perform well on an artificial test problem incorporating several different locations for multiple different activity types.

For agent based models focusing on the time allocation problem alone, the discrete parts of the activity planning problem can be neglected (i.e. activity pattern generation, location selection and mode choice). The simpler nature of the remaining optimization problem is exploited in Chapter 4 by using an implementation of the covariance matrix adaptation algorithm which is known to be efficient on continuous optimization problems. At the same time, the router, originally a separate module in our agent based model, is integrated into the activity planning module. The final model is sucessfully tested on the described simplified optimization problem showing the clear performance gain achievable using our approach.

Especially to predict short term reactions of the system to sudden changes (e.g. construction sites, sporting events, disaster etc.), within-day replanning capabilities are of interest. In Chapter 3, precomputation as a way of solving this problem is investigated. Activity planning with alternative execution paths is described as a reinforcement learning problem by using a discretized version of the marginal utility function. This problem is subsequently solved using the Q-Learning approach known from machine learning. In principle, the algorithm shows to solve the problem satisfactorily, but finding the resulting execution plan, including all alternative execution paths, represents a substantial overhead; this raises the question if such an approach would be efficient enough to be used for more than one million agents.

Overall efficiency is also one of the main aims in Chapter 4. To be efficient, reducing the number of iterations needed for the learning loop to converge to an equilibrium is of prime importance, as in every iteration the traffic flow microsimulation has to be run, which is very expensive. We show that changing the number of replanning agents from iteration to iteration similar to the method of successive averages can reduce dramatically the number of iterations needed for the agent learning to converge: 10 iterations of optimal learning can easily beat 100 iterations of replanning with constant replanning share.

9.2 Traffic Flow Microsimulation

Another way of improving the system's performance is by reducing the execution time for each iteration. As has been pointed out already, the traffic flow microsimulation has been traditionally responsible for most of an iteration's computational costs. In Chapters 5 and 6, exactly this fact is addressed.

First, a novel event-driven approach to the queue-based microsimulation problem is employed to save computation time during off-peak and highly congested times of day; this results in an at least ten times faster microsimulation without sacrificing any accuracy of the model. On the contrary, the models accuracy is improved by a notion of backwards traveling gaps producing better results in congested areas.

Second, this microsimulation model is parallelized using exactly the same dynamics as the sequential program. This parallel implementation showed to be very efficient: In our experiments on a 64 processor computer, the simulation ran 53 times faster than on a single CPU core. Because of the substantially better performance of the resulting microsimulation and consequently the reduced needs in terms of computing time, the traffic flow microsimulations fraction of overall computing time has turned from dominating to secondary.

To analyze the behavior of the introduced traffic flow microsimulation, the flow properties are investigated using several synthetic test scenarios in Chapter 7. It is shown that the microsimulation at hand has reasonable properties and reproduces a stylized fundamental diagram.

An extension of the microsimulation model is investigated in Chapter 8: The outflow capacities of the links are modulated by a time dependent green time fraction function which can be specified for each link individually. This gives the traffic planner more flexibility and naturally can produce better results when simulating cities.

Chapter 10

Further Work

To give some food for thought, this chapter contains a few ideas of new work packets or continuation of work presented in this thesis. This cannot nor wants to be a comprehensive list of what can or should be done. Rather, it is driven by conceptual and operational needs that arose during the writing of this thesis and work done by others collaborating on agent based traffic models in the Transport Planning group at ETH Zurich and in the Transport Systems Planning and Transport Telematics group at TU Berlin.

10.1 Solving the Location Choice Set Problem

In the daily planning process, location selection represents an important part, as it has influence on the number of trips executed by an agent and the distance traveled. The remaining problem is the enormous number of location alternatives available for each activity. For example, on the regional scale, there might be hundreds or even thousands of shops one of which has to be picked for each shopping activity. In terms of optimization, this represents a major challenge due to the enormous search-space size. There have been attempts to reduce this problem by preselecting a subset of these activities before the planning process starts thereby reducing the search-space size. Still, the number of locations remains substantial making location selection slow—maybe too slow for larger scenarios. A real solution to this problem would be an algorithm scaling sub-linearly with the number of alternative locations available. One possible approach to such an algorithm could be to exploit that far-off locations are of interest only if they provide a high quality, a quality which can be converted to utility by agents using this location. By making the visibility of activity locations depend on such a quality measure, many small facilities could be ignored in the location selection process which in turn would lead to better performance.

10.2 Finding an Efficient Within Day Replanning Approach

Apart from Chapter 3, this work deals only with daily activities planned ahead, not with reactions to unpredicted events during the day. This simplification is only a valid approximation if within day replanning effects are small compared to pre-planned behavior. Otherwise, we need new algorithms to explicitly cover changes to pre-planned behavior. Q-Learning, as presented in this work, certainly is a path to investigate, but this approach solves the problem by “foreseeing the unforeseen”, by thinking of all possible disturbances and computing the best reactions to all of them in advance. Quite clearly, this represents a big overhead in the case of normal daily plan execution. An algorithm avoiding this overhead would have to work “on the fly” by only issuing a within-day replanning if a certain trigger event, for example, a one hour delay compared to the pre-planned day, is detected during its execution. Such a replanning would need to be fast, as the traffic flow microsimulation is running by this time and would have to wait for the replanning to finish.

10.3 Adaptive Global Replanning Policy

For the overall performance of the presented agent based traffic model, the importance of the replanning share per iteration has been discussed and shown in Chapter 4. The results imply that big shares of agents should be replanning during early stages of the convergence process. During subsequent iterations, this number should be gradually reduced to finally arrive at replanning shares of about one percent to reach the best possible equilibrium at the end. The suggested replanning policy can be interpreted as optimal for the scenario given in Chapter 4, but it is not clear if this policy can be used equally with other scenarios. In other scenarios, the presented policy could turn out to be far from optimal which would badly impair the overall system's performance. To avoid this, rather an *adaptive* replanning strategy should be selected than a statically variable one. For this, a detector needs to be found to check if the current replanning share should be increased or reduced, respectively. Such a detector might even work on the agent level to give a per agent indication of the urgency to think a plan over. Such indicators might be the acquired utility of an executed plan as compared to its predicted utility.

10.4 Java Implementation of Traffic Flow Microsimulation

The algorithms presented in this work were all implemented in the C++ programming language. C++ is a powerful and modern programming language known for its good performance. Developing parallel programs in C++ is also easily possible. On the other hand, the MATSim project, which uses some of the work presented here, is being developed using Java mainly due to its better platform independence. The traffic flow microsimulation is being used as an extension module in that project due to its computational speed. Unfortunately, the data transfer from Java to C++ and back produces some overhead, and the desire has arisen for a native Java implementation of the same model including the parallelization as presented in Chapter 6. While, in theory, this task should not pose any unsolvable problem, it should not be underestimated. Simply translating the source code into Java would abandon the key advantage of an integrated traffic flow microsimulation in MATSim: the seamless integration into the rest of the project by using the same data model and therefore providing easy access to additional information if required.

Acknowledgments

Throughout my work I was surrounded by great people that supported me in my work, and in many cases also became good friends to me. Let me express my thankfulness to all of you.

I am much obliged to the two advisers of my thesis: Kai Nagel, thank you for introducing me into scientific work and for inspiring my interest in transport planning. Kay Axhausen, thank you for giving me the opportunity to continue my work at the IVT and for your enthusiasm for microscopic transport models.

I am also grateful to ETH Zurich for providing the facilities and the community that have formed a major part of my life in the past years.

Bibliography

Abraham, J. E. and J. D. Hunt (2002) Spatial market representations: Concepts and application to integrated planning models, paper presented at *the 49th Annual North American Meetings of the Regional Science Association International*, San Juan, November 2002.

AIMSUN (2006) AIMSUN, webpage, November 2006, `http://www.aimsun.com`.

Arentze, T. A., F. Hofman, H. Mourik and H. J. P. Timmermans (2000) Albatross: A multi-agent rule-based model of activity pattern decisions, *Transportation Research Record*, **1706**, 136–144.

Arentze, T. A. and H. J. P. Timmermans (2005) Representing mental maps and cognitive learning in micro-simulation models of activity-travel choice dynamics, *Transportation*, **32** (4) 321–340.

Arnott, R., A. de Palma and R. Lindsey (1993) A structural model of peak-period congestion: A traffic bottleneck with elastic demand, *The American Economic Review*, **83** (1) 161–179.

Axhausen, K. W. (1988) Eine ereignisorientierte Simulation von Aktivitätenketten zur Parkstandswahl, Ph.D. Thesis, University of Karlsruhe, Karlsruhe.

Balmer, M. (2007) Travel demand modeling for multi-agent traffic simulations: Algorithms and systems, Ph.D. Thesis, ETH Zurich, Zurich, May 2007.

Balmer, M., K. W. Axhausen and K. Nagel (2006a) An agent-based demand-modeling framework for large scale micro-simulations, paper presented at *the 85th Annual Meeting of the Transportation Research Board*, Washington, D.C., January 2006.

Balmer, M., K. W. Axhausen and K. Nagel (2006b) An agent-based demand-modeling framework for large scale micro-simulations, *Transportation Research Record*, **1985**, 125–134.

Barceló, J., J. L. Ferrer, D. Garcia, M. Florian and E. Le Saux (1998) Microscopic traffic simulation, in P. Marcotte and S. Nguyen (eds.) *Equilibrium and Advanced Transportation Modelling*, 1–26, Kluwer, Dordrecht.

Barrett, C. L., R. Jacob and M. Marathe (2000) Formal-language-constrained path problems, **30** (3) 809–837.

Beckman, R. J., K. A. Baggerly and M. D. McKay (1996) Creating synthetic baseline populations, *Transportation Research Part A: Policy and Practice*, **30** (6) 415–429.

Ben-Akiva, M. E., M. Bierlaire, H. Koutsopoulos and R. Mishalani (1998) DynaMIT: A simulation-based system for traffic prediction, paper presented at *DACCORS Short Term Forecasting Workshop*.

Ben-Akiva, M. E. and S. R. Lerman (1985) *Discrete Choice Analysis: Theory and Application to Travel Demand*, MIT Press, Cambridge.

Bernoulli, D. (1738) Specimen theoriae novae de mensura sortis, *Commentarii Academiae Scientiarum Imperialis Petropolitanae*, **5**, 175–192.

Bhat, C. R., J. Y. Guo, S. Srinivasan and A. Sivakumar (2004) A comprehensive econometric microsimulator for daily activity-travel patterns, *Transportation Research Record*, **1894**, 57–66.

Bowman, J. L. (1998) The day activity schedule approach to travel demand analysis, Ph.D. Thesis, Massachusetts Institute of Technology, Cambridge.

Bowman, J. L., M. A. Bradley, Y. Shiftan, T. K. Lawton and M. E. Ben-Akiva (1999) Demonstration of an activity based model system for portland, in H. Meersman, E. van de Voorde and W. Winkelmans (eds.) *World Transport Research*, 171–184, Pergamon.

Brilon, W., F. Huber, M. Schreckenberg and H. Wallentowitz (eds.) (1998) *Traffic and Mobility: Simulation—Economics—Environment*, Springer, Berlin.

Brilon, W. and N. Wu (1998) Evaluation of cellular automata for traffic flow simulation on freeway and urban streets, in W. Brilon, F. Huber, M. Schreckenberg and H. Wallentowitz (eds.) *Traffic and Mobility: Simulation—Economics—Environment*, 163–180, Springer, Berlin.

Cascetta, E. (1989) A stochastic process approach to the analysis of temporal dynamics in transportation networks, *Transportation Research Part B: Methodological*, **23** (1) 1–17.

Cayford, R., W.-H. Lin and C. F. Daganzo (1997) The NETCELL simulation package: Technical description, *Research Report*, **UCB-ITS-PRR-97-23**, California Partners for Advanced Transit and Highways (PATH), University of California, Berkeley, May 1997, `http://repositories.cdlib.org/its/path/reports/UCB-ITS-PRR-97-23`.

Cetin, N. (2005) Large-scale parallel graph-based simulations, Ph.D. Thesis, ETH Zurich, Zurich.

Cetin, N., A. Burri and K. Nagel (2003) A large-scale multi-agent traffic microsimulation based on queue model, paper presented at *the 3th Swiss Transport Research Conference*, Ascona, March 2003.

Chang, G.-L., T. Junchaya and A. J. Santiago (1994) A real-time network traffic simulation model for ATMS applications: Part I—simulation methodologies, *Journal of Intelligent Transportation Systems*, **1** (3) 227–241.

Charypar, D., K. W. Axhausen and K. Nagel (2006) Implementing activity-based models: Accelerating the replanning process of agents using an evolution strategy, paper presented at *the 11th International Conference on Travel Behaviour Research*

(IATBR), Kyoto, August 2006, `http://www.ivt.ethz.ch/vpl/publications/reports/ab387.pdf`.

Charypar, D., K. W. Axhausen and K. Nagel (2007a) An event-driven parallel queue-based microsimulation for large scale traffic scenarios, paper presented at *the 11th World Conference on Transportation Research*, Berkeley, June 2007, `http://www.ivt.ethz.ch/vpl/publications/reports/ab425.pdf`.

Charypar, D., K. W. Axhausen and K. Nagel (2007b) An event-driven queue-based traffic flow microsimulation, paper presented at *the 86th Annual Meeting of the Transportation Research Board*, Washington, D.C., January 2007.

Charypar, D., K. W. Axhausen and K. Nagel (2007c) An event-driven queue-based traffic flow microsimulation, *Transportation Research Record*, **2003**, 35–40.

Charypar, D. and K. Nagel (2003) Generating complete all-day activity plans with genetic algorithms, paper presented at *the 10th International Conference on Travel Behaviour Research (IATBR)*, Lucerne, August 2003.

Charypar, D. and K. Nagel (2005) Generating complete all-day activity plans with genetic algorithms, *Transportation*, **32** (4) 369–397.

Charypar, D. and K. Nagel (2006) Q-learning for flexible learning of daily activity plans, *Transportation Research Record*, **1935**, 163–169.

Chowdhury, D., L. Santen and A. Schadschneider (2000) Statistical physics of vehicular traffic and some related systems, *Physics Reports*, **329** (4–6) 199–329.

de Palma, A. and F. Marchal (2002) Real cases applications of the fully dynamic METROPOLIS tool-box: An advocacy for large-scale mesoscopic transportation systems, *Networks and Spatial Economics*, **2** (4) 347–369.

Doherty, S. T. and K. W. Axhausen (1998) The development of a unified modeling framework for the household activity-travel scheduling process, paper presented at *the 4th NECTAR Conference*, Tel Aviv, April 1998.

DynaMIT (2006) Intelligent transportation system program, webpage, `http://mit.edu/its/dynamit.html`.

DYNASMART (2003) DYNASMART, webpage, `http://www.dynasmart.com`.

DYNASMART (2006) DYNASMART, webpage, `http://www.dynasmart.com`.

Frick, M. and K. W. Axhausen (2004) Generating synthetic populations using ipf and monte carlo techniques: Some new results, paper presented at *the 4th Swiss Transport Research Conference*, Ascona, March 2004.

Graf, P. (2003) Simuliertes Lernen menschlicher Tagespläne mittels Methoden der künstlichen Intelligenz, Master Thesis, ICoS, ETH Zurich, Zurich, `http://e-collection.ethbib.ethz.ch/show?type=dipl&nr=132`.

Hansen, N. and S. Kern (2004) Evaluating the CMA evolution strategy on multimodal test functions, paper presented at *the Eighth International Conference on Parallel Problem Solving from Nature*, Birmingham, September 2004, `http://events.cs.bham.ac.uk/ppsn04/`.

Jara-Diaz, S. R. and R. Guerra (2003) Modelling activity duration and travel choice from a common microeconomic framework, paper presented at *the 10th International Conference on Travel Behaviour Research (IATBR)*, Lucerne, August 2003.

Karlström, A. (2004) A dynamic programming approach for the activity generation and scheduling problem, *Working Paper*, Transport and Location Analysis, , Stockholm.

Kaufman, D. E., K. E. Wunderlich and R. L. Smith (1991) An iterative routing/assignment method for anticipatory real-time route guidance, *Working Paper*, **91–02**, Department of Industrial and Operations Engineering, University of Michigan, Ann Arbor.

Kistler, D. (2004) Mental maps for mobility simulations of agents, Master Thesis, ICoS, ETH Zurich, Zurich.

Kitamura, R. (1996) Applications of models of activity behavior for activity based demand forecasting, paper presented at *the Activity-Based Travel Forecasting Conference*, June 1996, `http://tmip.fhwa.dot.gov/clearinghouse/docs/abtf`.

Mahut, M. (2000) A discrete flow model for dynamic network loading, Ph.D. Thesis, Département d'Informatique et de Recherche Opérationnelle, Université de Montréal, Montreal.

Marchal, F. (2001) Contribution to dynamic transportation models, Ph.D. Thesis, University of Cergy-Pontoise, Cergy-Pontoise.

Marchal, F. and K. Nagel (2005) Modeling location choice of secondary activities with a social network of cooperative agents, paper presented at *the 84th Annual Meeting of the Transportation Research Board*, Washington, D.C., January 2005.

MATSim-T (2004) Multi Agent Transportation Simulation Toolkit, webpage, `http://www.matsim.org`.

MATSim-T (2006) Multi Agent Transportation Simulation Toolkit, webpage, `http://www.matsim.org`.

Meister, K., M. Balmer and K. W. Axhausen (2005a) An improved replanning module for agent-based micro simulations of travel behavior, *Working Paper*, **303**, IVT, ETH Zurich, Zurich, `http://www.ivt.ethz.ch/vpl/publications/reports/ab303.pdf`.

Meister, K., M. Balmer, K. W. Axhausen and K. Nagel (2006) planomat: A comprehensive scheduler for a large-scale multi-agent transportation simulation, paper presented at *the 11th International Conference on Travel Behaviour Research (IATBR)*, Kyoto, August

2006, http://www.ivt.ethz.ch/vpl/publications/reports/ab388.pdf.

Meister, K., M. Frick and K. W. Axhausen (2005b) A GA-based household scheduler, paper presented at *the 84th Annual Meeting of the Transportation Research Board*, Washington, D.C., January 2005.

Miller, E. J. and M. Roorda (2003) A prototype model of 24-hour household activity scheduling for the Toronto area, *Transportation Research Record*, **1831**, 114–121.

MITSIM (2006) MITSIMLab, webpage, http://www.web.mit.edu/its/mitsimlab.html.

Nagel, K. and C. L. Barrett (1997) Using microsimulation feedback for trip adaptation for realistic traffic in dallas, *International Journal of Modern Physics C (IJMPC)*, **8** (3) 505–526.

Nagel, K. and M. Rickert (2001) Parallel implementation of the TRANSIMS micro-simulation, *Parallel Computing*, **58** (2) 1611–1639.

Nagel, K., D. E. Wolf, P. Wagner and P. M. Simon (1998) Two-lane traffic rules for cellular automata: A systematic approach, *Physical Review E*, **58** (2) 1611–1639.

Nökel, K. and M. Schmidt (2002) Parallel DYNEMO: Meso-scopic traffic flow simulation on large networks, *Networks and Spatial Economics*, **2** (4) 387–403.

Ortúzar, J. d. D. and L. G. Willumsen (2001) *Modelling Transport*, 3. edn., John Wiley & Sons, Chichester.

Pendyala, R. M. (2004) Phased Implementation of a Multimodal Activity-Based Travel Demand Modeling System in Florida. Vol. II: FAMOS Users guide, *Final Report*, Florida Department of Transportation, http://www.public.asu.edu/~rpendyal/FAMOSUsersGuide.pdf.

Raney, B. (2005) Learning framework for large-scale multi-agent simulations, Ph.D. Thesis, ETH Zurich, Zurich.

Recker, W. W. (1995) The household activity pattern problem: General formulation and solution, *Transportation Research Part B: Methodological*, **29** (1) 61–77.

Rickert, M. and K. Nagel (2001) Dynamic traffic assignment on parallel computers in TRANSIMS, *Future Generation Computer Systems*, **17** (5) 637–648.

Russell, S. J. and P. Norvig (1995) *Artificial Intelligence: a Modern Approach*, Prentice-Hall, uppersaddleriver.

Schwerdtfeger, T. (1984) DYNEMO: A model for the simulation of traffic flow in motorway networks, in J. Volmuller and R. Hamerslag (eds.) *Proceedings of the Ninth International Symposium on Transportation and Traffic Theory*, 65–87, VNU Science Press, Utrecht.

Sheffi, Y. (1985) *Urban Transportation Networks: Equilibrium Analysis with Mathematical Programming Methods*, Prentice-Hall, Englewood Cliffs.

TRANSIMS (2006) TRansportation ANalysis and SIMulation System, webpage, December 2006, `http://transims.tsasa.lanl.gov`.

VISSIM (2006) VISSIM, webpage, November 2006, `http://www.ptv.de/cgi-bin/traffic/graf_vissim.pl`.

Wardrop, J. G. (1952) *Some Theoretical Aspects of Road Traffic Research*, Institute of Civil Engineers.

Watkins, C. J. C. H. and P. Dayan (1992) Q-learning, *Machine Learning*, **8** (3–4) 279–292.

Yang, Q. (1997) A simulation laboratory for evaluation of dynamic traffic management systems, Ph.D. Thesis, Massachusetts Institute of Technology, Cambridge.