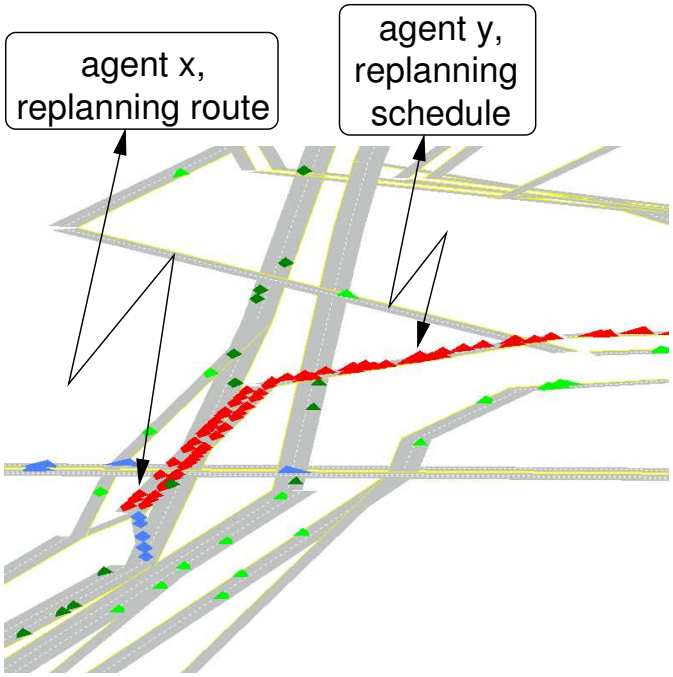
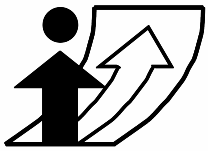


Computational methods for multi-agent simulations of travel behavior

Kai Nagel, TU Berlin
Fabrice Marchal, ETH Zürich

Resource paper
Workshop on Computational Techniques



Moving through nets:
The physical and social dimensions of travel
10th International Conference on Travel Behaviour Research
Lucerne, 10–14. August 2003

Computational methods for multi-agent simulations of travel behavior

Kai Nagel
Inst. for Land and Sea Transport Systems
TU Berlin Sek SG12
D-10587 Berlin
Germany

Phone: +49 (0)30 314 23308
Fax: +49 (0)30 314 26269
eMail: nagel@vsp.tu-berlin.de

Fabrice Marchal
Computational Laboratory (CoLab)
ETH Zürich
CH-8092 Zürich, Switzerland

Phone: +41 (0)1 632 56 79
eMail: marchal@inf.ethz.ch

Abstract

Travel behavior research is, by definition, concerned with the behavior of travelers. In order to make those research results useful for policy decisions, often the behavior of many travelers needs to be considered in conjunction. This can either be achieved by models which look at groups of travelers, or by models which look at individual travelers. This paper is concerned with the latter, and how such models of individual travel behavior research can be combined into a model-based transportation forecasting tool. The method to achieve this is called multi-agent simulation.

This paper looks at multi-agent simulation for travel behavior research first from a modeling and then from a computational perspective. The modeling part discusses the main issues: simulation of vehicles and travelers, strategy generation, learning/feedback, and initial/boundary conditions. The computational part then investigates how those modeling approaches can be implemented on existing computer architectures. Special emphasis is put on the interoperability between modules coming from different authors, and on parallel and distributed computing in order to achieve computability of large scenarios.

The strong emphasis on the computational aspects is related to the belief that modeling and implementation are not independent: Certain models are easier to implement than others; and most implementation decisions eventually favor some models over others.

Keywords

multi-agent simulation; agent-based modeling; traffic simulation; parallel computing; distributed computing; travel behavior International Conference on Travel Behaviour Research, IATBR

Contents

1 Introduction **2**

2 Multi-agent simulation modeling issues **3**

2.1 Initial and boundary conditions 5

2.2 The MobSim 5

2.3 Strategy generation 6

2.4 Adaptation, learning and feedback 7

2.5 Backward compatibility to 4-step process 10

2.6 Summary 13

3 Computational techniques **14**

3.1 Introduction 14

3.2 Coding 14

3.3 Search methods 18

3.4 Databases for initial/boundary conditions 20

3.5 Module coupling 24

3.6 Parallel computing 28

3.7 Distributed computing 35

3.8 Other issues 39

4 Summary **40**

1. Introduction

Human society is quite generally interested in forecasts of all kinds. This concerns both the short-term, e.g. weather, and the long-term e.g. climate. Reasons for such an interest are for instance economic interests, or simple curiosity. There is also an interest in forecasting traffic, both in the short-term, e.g. for daily operations, and in the long-term, e.g. for regional and urban planning.

Forecasts are based on models. A model can for example be human intuition, or a linear extrapolation. For complex problems, such as transportation, there is a general agreement that at least some of the forecasts should be based on solid scientific and engineering methods.

Regarding these scientific and engineering methods for forecasting, two broad trends can be distinguished: "black box" models, and "models from first principles". Black box models refer to methods which are not geared to a particular problem, but rather general in their applicability. Examples are linear extrapolation or neural nets. Common to these models is that they do not look at the microscopic mechanics of a system. In contrast, models from first principles look at those microscopic processes and attempt to exploit them. An example is the theory of fluid-dynamics, which can be justified by an elaborate derivation from kinetic equations. Models from first principles are, at least in theory, more powerful than black box methods, since they are geared to the specific application. As a side effect, they normally allow more insight and understanding of the system under consideration. Their disadvantage is that they are more costly and difficult to develop.

Clearly, there are all kinds of intermediate approaches. For example, sometimes a linear law is based on first principles (e.g. the ideal gas equation); or a method relies on *some* understanding of the system, without looking at *all* the microscopic details.

This contribution will focus on microscopic models for traffic forecasting. The term microscopic means that the individual traveler is the principal unit of modeling. This is in contrast to other approaches, most prominently the four step process, which look at aggregated streams of travelers rather than individual travelers. In agreement with other areas of computational science, the microscopic approach will be called a **multi-agent simulation (MASim)**. The difference between an agent and a (physical) particle is the fact that the agent can have an internal state, internal processing, etc. This means that two outwardly identical agents, when submitted to exactly the same situation, can make fundamentally different decisions. Clearly, there are again all kinds of intermediate models between pure physical particles and truly autonomous, intelligent agents.

The paper consists of two main parts, one addressing modeling issues and the other computational issues. The part on modeling issues (Sec. 2) is rather short, concentrating on initial/boundary conditions (Sec. 2.1), simulation of the physical reality (Sec. 2.2), strategy generation (Sec. 2.3), and learning/feedback (Sec. 2.4). This is complemented by a section on backward compatibility to the 4-step process (Sec. 2.5), which discusses how modules from agent-based approaches can be coupled to modules from the 4-step process.

The part on computational techniques (Sec. 3) concentrates on how simulations can be implemented. It starts with the issues of coding (Sec. 3.2), a very short look at computer science and computational science search algorithms (Sec. 3.3), and a discussion of database applications in the maintenance of initial and boundary conditions (Sec. 3.4). Arguably, the most important and least resolved computational issue is the interoperability between modules (Sec. 3.5). In contrast, parallel computing, which can be used to speed up individual modules, at this point can be considered a mature technology, even if practical applications are still somewhat rare (Sec. 3.6). Sec. 3.7 then discusses how the issues of interoperability and of parallel computing interact, leading to the very active fields of Distributed Artificial Intelligence, Software Agents, and Peer-To-Peer Systems, but also to a large number of open issues and few established and mature solutions. The last section in the computational part (Sec. 3.8) covers issues such as the emerging GRID technology and the use of visualization and virtual reality.

The focus on the computational techniques is borne by our beliefs that modeling and implementation are not independent. Rather, our experience is that certain models are easier to implement than others, and certain implementations favor certain models over others. In other words, there will be a tendency

to select models which are easy to implement over models which may be more realistic, and such a pragmatic approach may even be justified by the results. Moreover, once a particular implementation has been selected, there will be a certain amount of *lock-in* toward certain models until someone else takes it on to start a competing, different implementation. The paper concludes by a short discussion and a summary.

2. Multi-agent simulation modeling issues

Constructing a multi-agent simulation is, at least initially, a rather straightforward process: One takes some simulation substrate, e.g. a 2-dimensional plane or a road network, populates it with agents, and gives them rules how they behave. Since the intention is to simulate the real world rather than some artificial system, much of the information can be taken from the real world.

It may be worth noting that this approach is usually rather obvious to people with a background in physics, engineering or computer science where many models involve individual interacting units. On the opposite, it is often counter-intuitive for people with a background in economics. One reason is that economists are trained to think about the economic system in terms of aggregated quantities, such as demand and supply, which are decoupled from individual actors.

As the ideal gas law in physics demonstrates, these views are not incongruous; in fact, much of the progress in physics in recent years is owed to the derivation of macroscopic laws from microscopic rules. Yet, it is our feeling that much of economic theory including travel demand forecasting is not accustomed to this microscopic view.

The advantage of the microscopic, agent-based view is that, at least in principle, it can be made arbitrarily realistic. One can start with rather simple models and simulations, and every time some effect is not captured, one can add more details and thus eventually capture the desired effect. Nevertheless, there are obvious limits to this: Coding all these details takes time and effort; the knowledge, for example about human behavior, may not be sufficient; the necessary input data for all the details, for example certain demographic characteristics, may not be available.

The main purpose of the microscopic description in multi-agents system is to reveal and to forecast properties of the system that cannot be deduced only from a model of the physical system or from a set of assumptions about human behavior. Indeed, the simulations aim to bring explanations that arise from the interaction between a physical environment and the decisions of agents regarding their actions in that environment. Also, because of the complexity of the physical environment, the interaction between the agents is not trivial and cannot be expected to be modeled by a simple law (e.g. congestion in traffic networks). Therefore, multi-agent simulations of systems with a physical reality generically consist of at least two components (Fig. 1):

- The component(s) which compute(s) the physical aspects of the system (such as excluded volume, limits on acceleration, etc.). This simulation of mobile particles will be called the **MobSim** throughout this text (see Sec. 2.2). The particles in this context can be vehicles (potentially occupied by several individuals) or single individuals (such as pedestrians).
- The component(s) which compute(s) the strategies of the agents. This includes for example mode/route choice, activity scheduling, reaction to congestion, choice of residence, choice of workplace, etc. (see Sec. 2.3).

The distinction between the strategy generation and the MobSim is made because they require very different sets of tools, and also often different skills from the modelers. For transportation applications, one needs to make both components useful for the real world. This can, in our view, best be achieved by completely separating the strategy generation from the MobSim. Then, strategies are submitted to the MobSim, which executes them and returns the strategies' performance. The agents can react to this information, as discussed in Sec. 2.4 on learning, .

Figure 1: Strategic layer, physical layer. Note that a traffic management center is just another strategic entity that communicates with entities in the physical system (such as adaptive traffic lights). Source: adapted from (Ferber, 1999, Chapter 4)

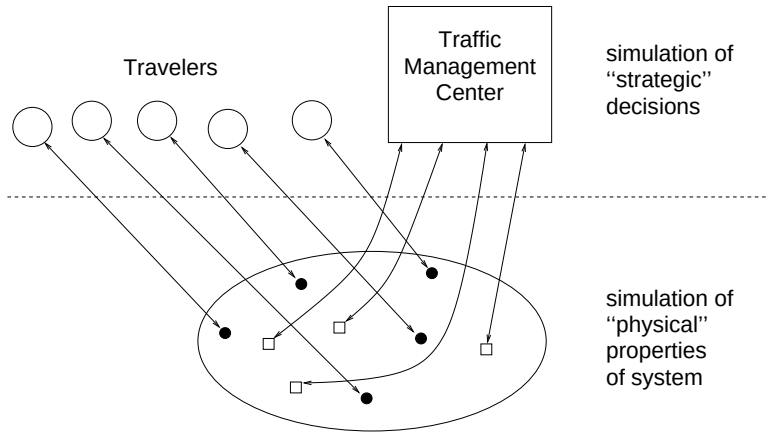
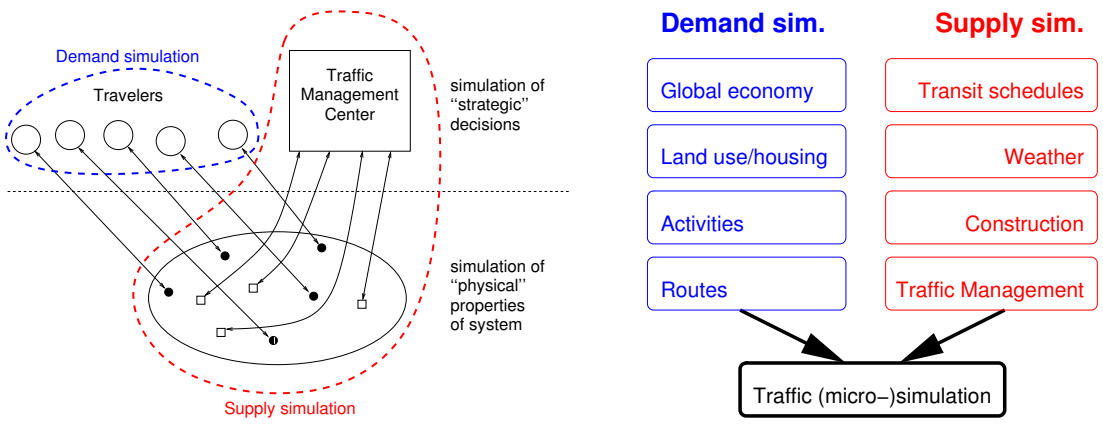


Figure 2: Strategic/physical layer vs. demand/supply simulation.



Such a setup still gives no guarantee that the MobSim has any relation to the real world. It is, however, now possible to construct the MobSim with principles borrowed from the natural and engineering sciences, where there is much more experience with the simulation of realistic systems. In contrast, the strategy generation modules can be designed with principles from Artificial Intelligence and/or Psychology.

Fig. 2 shows how the strategic/physical level approach relates to the often used demand/supply dichotomy. The typical inclusion of the MobSim into the supply side leads to a rather asymmetric picture (Fig. 3(a)). An alternative interpretation is to reduce "demand" and "supply" to their strategic dimensions, and let the simulation of the physical level be the neutral "referee" or "moderator" that calculates the consequences of the demand and supply strategies. This leads to a much more symmetric picture (Fig. 3(b)). In this interpretation, the *decision* to build a road would be a strategic decision on the supply side. But once the road is built, it becomes a part of the physical environment. It should be noted that such an approach is not inconsistent with trends in economics, where it is increasingly noted that, between demand and supply, there is a market with its own mechanisms and mechanics, which need to be

explicitly modeled for many aspects of economics (Shubik, personal communication).

Besides the MobSim and the strategy generation, there are two more components which are necessary to make the whole simulation work: a method to do learning/feedback and to fix initial/boundary conditions:

- A *complete* strategy (in the sense of game theory) would be one that contains all the responses to all possible conditions that a traveler could encounter during the run of the MobSim. In general, it is neither computationally possible nor behaviorally realistic to compute such a complete strategy. Instead, one typically implements a **learning mechanism**, by which the agents learn to modify and improve their strategies. An example for this is the reaction to congestion: Agents make plans, the MobSim execute them simultaneously, agents revise their plans, these plans are executed again, etc. Issues related to learning will be treated in Sec. 2.4.
- **Boundary conditions** refers to the data that remain fixed throughout the simulation, such as the road network. **Initial conditions** define how the simulation is started (see Sec. 2.1).

2.1 Initial and boundary conditions

Simulations need boundary and initial conditions. Boundary conditions are the data that does not change during the simulation, such as (possibly) the transportation system, the demographic characteristics of the population or the land use. Similarly, a simulation needs to be started somehow, which provides the initial conditions. For example, one could start with a certain population and then evolve it through the course of the simulation.

In the context of multi-agent travel behavior simulations, the most important pieces of data are:

- **Physical layer**. This refers to the nodes and links of the road network, but also to information about public transit, about pedestrian travel possibilities, etc.
- **Census data** or **synthetic population data**. The population of agents used by the multi-agent simulation can be provided by census records or by a synthetic population generation module.
- **Land use data**. An important aspect of activity-based demand generation is the location choice for activities, since activities at different locations are connected by travel. These data consist of the location of facilities and their characteristics (e.g. opening hours and capacity).
- **Study configurations**. For each simulation run, there needs to be configuration files that define the run. This includes the scenario files that are used, but also simulation parameters such as the spatial resolution of the simulation.

The storing of **survey data** is not part of this text; this is treated in other sessions of the conference. However, micro-simulations of travel behavior clearly depend on behavioral parameters that are obtained from the evaluation of surveys. For the purposes of this text it will be assumed that such parameters are obtained from elsewhere, and that they are stored as part of the synthetic population or as part of the study configuration.

2.2 The MobSim

As said above, the MobSim refers to the simulation of the physical transportation system. It computes what happens to the agents' strategies when they are confronted with (a synthetic version of) the real physical world. With respect to a conference on travel behavior research, the MobSim is maybe not of prime importance. Yet, there is an increasing number of voices arguing that issues of "artificial intelligence" cannot be decoupled from the physical system. This "embodiment hypothesis" means, for travel behavior, that any results from a travel behavior model need to be submitted to some version of the real world to make sense. Typical issues that become clear when submitting strategies to such a "real world

evaluation" are that the strategy could have been incomplete, or that the execution under real physical constraints leads to a much different behavior than anticipated.

The extreme of this would be to have some robotic lab where miniature travelers move through a toy traffic system. Short of this, one can use a simulation of the physical system as a proxy for embodiment. If, as suggested above, the simulation of the physical system is programmed by a totally different team than the simulation of the strategy generation, then there is some hope that the strategy will also make sense in the real world.

Beyond these aspects, a precise discussion of MobSim techniques is not critical to this paper. Suffice it to say that it is now possible to write virtual reality simulation systems, where the analyst can either look at a virtual reality version of the world as an independent observer, or he/she can participate as a traveler, e.g. as a driver. Yet, although such simulation systems are feasible, for many investigations they are too slow, too costly, and to extensive to program and to maintain. In such situations, it is often possible to use a much simpler method. In order to maintain the multi-agent approach, it is possible to have each agent represented individually in such a simulation, but apart from that many details from reality can be neglected and a useful result can still be obtained. Examples of such simulations in the transportation field are DYNEMO (Schwerdtfeger, 1987), DYNAMIT (DYNAMIT www page, accessed 2005), DYNASMART (DYNASMART www page, accessed 2005), or the queue model (Gawron, 1998b,a).

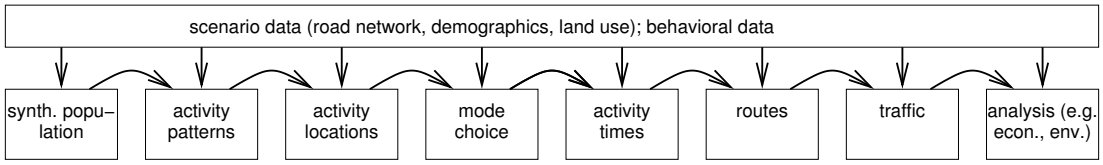
2.3 Strategy generation

The MobSim computes the physical aspects of movement, such as limits on capacity, storage, or speed. In particular it computes the aspects of interaction, such as congestion. The MobSim needs information about where travelers enter and leave the network, which turns travelers take at intersections, etc. As mentioned earlier, these aspects can be called plans, or strategies. For the transportation simulation, this means that travelers know where they are going, when they want to be there, and the route they want to take to get there. This kind of strategic knowledge is in stark contrast to, say, the simulation of ants in an ant-hill. It also makes the simulation design considerably more demanding, since the generation and handling of strategies is a whole problem of its own. Our own approach to this problem, as said before, is to allow a distributed design, that is, MobSim and strategy generation should be separated as much as possible, and in fact we also intend to have more than one strategy generation module in the future. This is further discussed in Sec. 3.7.

Important strategy generation modules for transportation applications are route generation, activity generation, car ownership models, housing choice, commercial location choice, land use changes, etc. Such models and modules are discussed in much breadth and depth at other places in this conference. Integrating such models into a multi-agent simulation framework is in principle once more straightforward. In practice, however, there are several issues that need to be considered:

- The unit of analysis needs to be consistent. The typical unit of analysis in a multi-agent travel behavior simulation is the traveler; an alternative would be the household. Once all modules agree on the unit of analysis, it is relatively easy to treat information consistently. If modules use different units of analysis, then information will get lost in the conversion. For example, during a conversion of agent-based activity chains to an origin-destination matrix one will lose any information about effects along the time axis (a person running late in the morning may carry the delay through the day), and one will lose any connection to demographic characteristics.
- There needs to be some standardization of what each module expects. For example, there is now some agreement that a typical daily plan consists of an activity schedule, consisting of a sequence of activity types, together with location and time information. Furthermore, activities at different locations are connected by trips, for which information such as mode and route is given. As long as there is essential agreements on the hierarchy of these elements, different modules can work together.
- Computational issues. These will be discussed in Sec. 3.

Figure 3: Some modules of an activity-based travel behavior simulation system.



Some modules of an activity-based travel simulation system are shown in Fig. 3. It is plausible to assume that, from left to right, each module adds more specific agent data. For example:

- The population generation module determines age, gender, income, and home location (e.g. Beckman *et al.*, 1996).
- The activity pattern module adds an activity pattern, such as home-work-shop-home.
- The activity location module locates those activities.
- The mode choice module determines the mode of transport.
- The activity timing module decides how long each activity takes, and when the daily activity plan is started, and adds that timing information.
- The route choice module determines the exact route.
- Finally, the MobSim executes all plans of all agents simultaneously according to the specifications.

This is just an example of such a simulation system. Modules can be combined (such as all the activity-generation modules (e.g. Arentze *et al.*, 2000; Kitamura, 1996; Bowman, 1998)) or decomposed (such as primary and secondary activity location choice), and other modules will have to be added (such as car ownership). Nevertheless, it should be visible from the example how each individual module *adds* information, thus making the agent strategy more and more complete from one module to the next.

The long-term goal here is to define data exchanges so that these systems can work together. In the past, the famous origin-destination (OD) matrices had exactly that effect: Demand generation modules knew what output to construct, and assignment modules knew what input to expect. It is both possible and necessary that agent-based simulation packages will reach the same level of interoperability.

2.4 Adaptation, learning and feedback

2.4.1 Introduction

Above, in particular in Fig. 3, it was implied that the generation of agents' strategies is a linear process, going from synthetic population via activities to route choice. It is probably clear to everybody that in practice there is also backward causality. For example, congestion is the result of (the execution of) plans, but plans are based on (the anticipation of) congestion.

Therefore, feedback mechanisms have to link the different modules. Multi-agent systems intend to implement these mechanisms by mimicking the actual response of human agents, not by simply linking modules artificially. Unfortunately, most of the methodology to model these reactions is missing.

2.4.2 General issues of learning

Learning is not included in one of the mainstays of travel behavior research, the random utility model (Ben-Akiva and Lerman, 1985). How does learning enter the picture? Laying out a theory of learning agents is beyond the scope of this paper, but some aspects are worth noting:

- First, it makes sense to distinguish between single-agent learning, and system-wide co-evolution. **Single-agent learning** concerns a single agent when faced with the environment. In order to make scientific progress, it makes sense to assume that environment as stationary, i.e. that some underlying statistical distributions do not change over time. In contrast, **system-wide co-evolution** is concerned with the dynamics of a system in which many agents learn simultaneously.

- Within single-agent behavior, one can differentiate between behavior based on a scoring function (utility, fitness, prospect theory, etc.) and behavior based on symbol processing (for example following signs in an airport). There is in fact a related dichotomy between utility-based models and process models in activity scheduling.

- For single-agent behavior based on a scoring function, one can either assume that the agent is capable of "constructing" the optimal solution, or that the agent attempts to improve the value of this function from one period to the next. If the improvement algorithm converges to the optimal solution, and if one is just interested in the converged result, then constructing the solution is just a short-cut for learning it. "Fuzzifying" the choice, for example via the well-known $p_i \propto e^{\beta U_i}$, does not change the principal argument. These remarks show, at least intuitively, the connection between random utility theory and single-agent evolutionary modeling: Under certain conditions, the single-agent evolutionary algorithm will converge to the random utility result.

  If, however, one is interested in the transients, or if one assumes that an agent eventually stops improving even if the best solution is not reached, then the two methods are no longer equivalent.

- When many agents learn simultaneously, one obtains a co-evolutionary dynamic system. It does not make a fundamental difference if agents are capable of constructing the optimal solution ("best reply") or if their behavior is given by some other rule: Since all agents co-evolve simultaneously, a strategy that was a best reply at some period is in general not a best reply at some other time. In consequence, even agents that individually follow random utility theory or some other best reply behavior will still generate a complicated co-evolutionary learning system.

  Such a co-evolutionary learning system is a dynamical system. Depending on its exact properties (deterministic or stochastic, Markovian or not, etc.), it can display a large range of behaviors, for example (**?**)e.g.][]Hofb:Sigm:book, Cantarella:Cascetta, Bottom:thesis,Watling:stochastic: convergence to a fixed point, convergence to a periodic attractor, chaos, convergence to fixed probabilities in phase space, ergodicity, broken ergodicity, etc.

- It makes some sense to look at how this problem is approached traditionally. The traditional concept is the **Nash Equilibrium (NE)**. As is well known, a system is at a Nash Equilibrium if no agent can improve by unilaterally changing its strategy. In the context of what has been said above, this means that all agents are using a best reply.

  The Nash Equilibrium concept is related to the co-evolutionary system by the fact that, if "round-robin" single-agent learning converges to a individually optimal solutions, then a fixed point of the co-evolutionary system is a Nash Equilibrium (Hofbauer and Sigmund, 1998). However, remember that the co-evolutionary learning system can display a much richer variety of behaviors than just convergence to a fixed point.

- It is interesting to note that, although the NE is a normative concept (the system *is* at a NE if ...), most if not all algorithms to actually find a NE in static assignment are iterative. That is, they have some resemblance to iterative learning, except that the iterations are set up to fulfill some mathematical property, such as guaranteed convergence, which means that they explicitly *do not* model human behavior.

- As a last point, let us mention within-day replanning. The idea of within-day replanning is easy to formulate: If an agent encounters something unexpected during the day (such as unexpected congestion, or an ice-cream stand during a warm day), the agent should be able to adapt its plan

immediately (Axhausen, 1990; Doherty and Axhausen, 1998). This is in principle straightforward to implement, although there are some performance issues for large scale scenarios and parallel implementations (Sec. 3.7). It is, however, important to note that such an implementation is not automatically consistent with evolutionary game theory, which is the basis for nearly all theory of co-evolutionary dynamics that we know.

This problem has to do with timescales of adaptation: If a traffic management center announces tolls (slow adaptation) and travelers react to the announcement (fast adaptation), then the result may be different from a traffic management center that adapts tolls within minutes (fast adaptation) and without announcement while travelers can only react to some average toll they encounter over the course of several trials. In static game theory, the problem is known as sequential games, multi-stage games, or Stackelberg games (e.g. Zuylen and Taale, 2004).

After this tour d'horizon through some aspects of feedback within agent-based behavioral models, let us look at some specific implementations. The purpose of presenting those is to anchor the discussion about computational methods in what is possible today, and then look from there to what may be possible in the future.

2.4.3 "Basic" agent-based feedback

A possible version to model learning is as follows:

1. The system begins with an initial set of strategies, one per agent, based on free speed travel times, which represent a network without congestion.
2. The MobSim is executed with the current set of strategies.
3. Some fraction f of the population requests new strategies which replace the old strategies.
4. This cycle (i.e. steps 2 through 3) is repeated until some kind of stopping criterion is reached.

Many variations of this are possible (Kaufman *et al.*, 1991; Lohse, 1997; Boyce *et al.*, 1997). For example, the new strategies could be best replies to the previous iteration or to some average of the previous iterations. The latter can for example be obtained from exponential smoothing, i.e. $\overline{T}(n) = \alpha T(n) + (1-\alpha)\,\overline{T}(n-1)$, where n is the iteration index, $\overline{T}$ is the value that is used for any strategy computations, and $0 < \alpha \leq 1$ determines how much "new" information T is included in each iteration. If is also possible to follow random utility theory and to chose between many options based on, say, $e^{\beta\, U_i}$.

According to what was said earlier, the choice of the learning models and their free parameters interacts with the behavior that the system will display. For example, a small replanning fraction f, averaging over previous iterations, or some small value of α either force the system to a fixed point or to relatively small cycles. Doing the opposite leads to cycles or possibly to chaotic behavior. Yet, one does not want to select f and/or α too small since then relaxation of the system toward the attractor (if this is what is desired) will be too slow. Heuristically, combining, in each iteration, 90% of "old" information with 10% of "new" information (i.e. $f = 0.1$ or $\alpha = 0.1$) seems to work well.

2.4.4 Multiple strategies per agent

A "best reply" strategy means that a replanning agent can construct the optimal response to a given situation of the environment. An example is the selection of a shortest path in a time-dependent network. In complex systems, the assumption that agents perform such strategy selection seems unreasonable because it contradicts the cognitive ability of humans: travelers cannot discriminate, following the same example, two paths that differ by one second. Nevertheless, best reply strategies are appealing because we can borrow much from Operations Research and Computer Science to implement the decision model. The approach of random utility theory relaxes these assumptions by assigning higher probabilities to

strategies with higher outcomes. By doing so, similar alternatives will be selected with similar probabilities. This theory has two main drawbacks: firstly, there is always a finite (though small) probability that agents are going to select a very low utility option; secondly, it is based on the assumption that users face or are aware of all the potential outcomes, which is also not realistic from the behavioral point of view. An alternative to these approaches is to borrow a method from Complex Adaptive Systems, where possible solutions are permanently added and removed. The system then choses between those solutions with probabilities that may resemble random utility theory. An implementation of this may look as follows (Raney and Nagel, 2002, 2003):

1. The system begins with an initial set of strategies, one per agent, and the MobSim is executed with those strategies.
2. Each agent measures the performance of his/her strategy based on the outcome of the simulation. In contrast to the basic approach, this information is memorized (stored), associated with the strategy that was used.
3. A fraction of the population requests new strategies from some arbitrary external strategy generation module. Agents which request/receive new strategies select those for execution.
4. Agents who did not request new strategies choose a previously tried strategy, by comparing the performance values for the different strategies. For example, they use a multinomial logit model $p_i \propto e^{\beta U_i}$ for the probability p_i to select strategy i, where U_i is the utility (score) of option i and β is an empirical constant.
5. The MobSim is executed with the selected strategies.
6. This cycle (i.e. steps 2 through 5) is repeated until some stopping criterion is met.

An advantage of simulating the memory of agents is that the overall system is more realistic and in fact more stable from the numerical point of view. It also has the advantage to relax considerably the design constraints of the modules that generate the strategies. These modules can produce strategies that are suboptimal and in limited number. They will be evaluated by the agents that perform their own trial-and-error process. It is especially useful when the constraints put on a deterministic algorithm would not be met (e.g. selection of the best path with multiple criteria like minimum delay and maximum sight-seeing). An important drawback, though, is that the learning process by trial-and-error can be extremely slow (i.e. agents would need several lifetimes to build a realistic choice set). An obvious answer to this problem is to model also the exchange of information between agents. Again, models are missing there and research efforts are needed into that direction.

The "multiple strategies" approach brings our agent-based simulation closer to a classifier system as known for Complex Adaptive Systems (Stein and others, several volumes, since 1988). From here, alternative methods of agent learning, such as Machine Learning or Artificial Intelligence, can be explored. Similarly, one can explore different assumptions about the cognitive abilities of the human agent: How far do agents remember (e.g. Markov process of order n)? How much do they value recent experiences? Is there some risk evaluation or are the outcomes considered as deterministic? How far do agents anticipate the future outcome of the system (under/over shooting)?

2.5 Backward compatibility to 4-step process

So far, this paper has discussed multi-agent simulations for travel behavior simulation without much connection to established methodologies, notably the 4-step process (e.g. Sheffi, 1985; Ortúzar and Willumsen, 1995). Yet, given the large investments, in terms of data, training, and software, of some metropolitan planning organizations into the 4-step process, it is unlikely and also undesirable to abandon those investments and to start over with an agent-based approach. It is therefore important to discuss possible transition paths.

Fortunately, such a transition path is possible, and it is in fact possible to obtain meaningful results on every step along the way.

2.5.1 From agent-based to 4-step

The first part of the argument consists of the observation that it is always possible to obtain 4-step process inputs and outputs from the agent-based approach: It is possible to aggregate the results of the activity generation into OD matrices; it is possible to separate those OD matrices by mode; it is possible to obtain those OD matrices for any desired time slice, including hourly, AM/PM peak, or daily; and it is possible to obtain link volumes or link travel times from the MobSim.

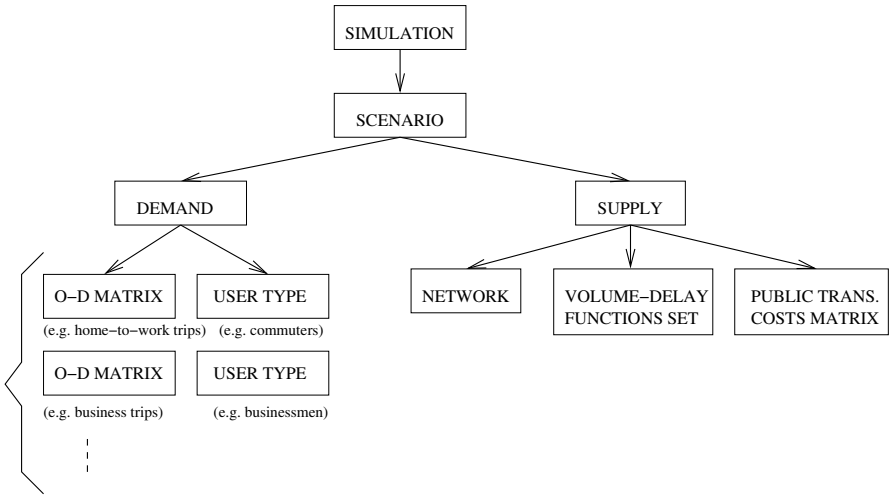
This makes it possible, for example, to use activity-based demand generation and to feed it into the route assignment part of the 4-step process, a path that is indeed in the process of being implemented in several cities (e.g. Portland/OR, San Francisco, Ohio). A possible disadvantage of this approach is that the information that is available at the activity-generation level, e.g. demographic data or tight activity chains, will get lost in the process: After the activity chain is translated into trips, it is perfectly possible that a person leaves an activity location before it arrives!

A more troubling problem is that it does not really make sense to feed time-dependent activity chains into a static assignment model. A short answer is to use dynamic traffic assignment (DTA) models instead and to feed them with time-dependent OD matrices. However, this leads to a discussion with many facets, and discussing all of them in detail goes beyond the scope of this paper. Some of these issues are listed here:

- At the minimum, separate assignments are run for the important periods of the day (e.g. early morning, AM peak, off-peak, etc.). Alternatively, some assignment models have an explicit time-dependent mechanism in which they move unfinished trips forward into the next time slice (e.g. Kaufman *et al.*, 1991; Friedrich *et al.*, 2000).
- Most of the DTA models suffer from the same shortcoming as static assignment models, i.e. that their representation of congested traffic including queue built-up is wrong since they do not represent the congested branch of the fundamental diagram. In addition, while static assignment models have the very nice "uniqueness" feature (under certain conditions), much less is known about the mathematical features of the dynamic versions.
- There is now increasing awareness of the fact that the congested branch of the fundamental diagram should be represented correctly in what is now called the "network loading model", and several models are implementing space constraints and queue built-up ("physical queues") (Schwerdtfeger, 1987; DYNAMIT www page, accessed 2005; DYNASMART www page, accessed 2005; Gawron, 1998a; Astarita *et al.*, 2001). In general, this requires to disaggregate the OD matrix into individual trips. Mathematical uniqueness properties of Nash Equilibrium assignment almost certainly do not exist with such models (Daganzo, 1998).
- DTA models are mainly concerned about short-term operational predictions and therefore have focused on the detailed description of congestion and vehicle movements. In doing so, the computational overhead prevents most of them of modeling the city-wide scale required by transportation planning.
- DTA models often calibrate "raw" dynamic OD matrices against traffic counts, potentially in real-time. Unfortunately, this breaks the feedback loop between agent decisions and system performance since there is no behavior explanation to the time-dependency of the input flows: Trips in the OD matrix are changed because the emergent result does not match the observation, rather than because of some behavioral reaction. This is a typical case where the "time series analysis/black box" approach for practical operations conflicts with the idea of building of model from first principles.

  In other words: DTA models may be a much better starting point for dynamic network loading models than stepwise extensions of the 4-step volume-based link cost function. However, with DTA models one needs to be careful that one only uses the pieces that are behaviorally justified while omitting those which are just there to make the DTA models operational in a real-time real-world context.
- Finally, most of the models mentioned above do not represent flexible agent features such as dependencies along the time axis, or replanning during the day.

Figure 4: The data sets hierarchy of METROPOLIS. The demand is segmented in an arbitrary number of OD matrix/User type pairs to capture trips with different purposes or morning and evening peaks.



- It is worth mentioning that network loading models with physical queues can be made completely agent-based without needing additional input data. That is, the early perception that agent-based traffic simulation models are always extremely data hungry is *not* correct.
- If, however, a more realistic representation of the traffic dynamics is desired, then a more realistic network loading model can easily be used for any of the approaches.

A somewhat different path is followed by METROPOLIS (de Palma and Marchal, 2002, 2001), a dynamic traffic simulator that is partly agent-based: travelers with individual characteristics perform mode, departure time and route choice in a time-dependent context. Instead of making the demand generation time-dependent by going all the way to activity-based demand generation, it takes instead the conventional morning or afternoon peak period OD matrix as a starting point, and then self-consistently generates the whole temporal structure of the peak period. This is done by assuming that trips going to certain destinations (such as where workplaces are) have certain time constraints, and then applying a model for departure time choice for those trips (see Fig. 4). This approach allows to capture time-dependent trade-off between congestion and activity schedules constraints for the morning or the evening peak. However, it does not provide a complete description of daily tours and their schedules as the outcome of individual choices.

A big advantage of this approach is that it is more "agent-based" than standard assignment but does not need the full agent-based demand generation or any other procedure to generate time-dependent OD matrices. Our own practical experience suggests that, given current technology and implementation status, applying METROPOLIS to a new scenario takes a small number of days, while applying our agent-based approach on a new scenario takes several months.

2.5.2 From 4-step to agent-based models

The reverse journey, i.e. obtaining agent-based data from the 4-step process, is less straightforward. The problem is that when going from agent-based to 4-step, information gets lost. This implies that in the reverse direction, from 4-step to agent-based, some information needs to be generated, or a decision needs to be made to ignore it. For example, it is perfectly possible to go from OD matrices to activity-based plans by just making up a virtual person for each trip in the OD matrix. However, in contrast to the fully agent-based approach, information about trip chaining will get lost, i.e. a delay in the morning, which may cause further delays throughout the day in the agent-based approach, will not have any consequences once converted into an OD matrix.

2.5.3 Theoretical justification of aggregated models

From a theoretical perspective, it is clear that the fully agent-based approach has many advantages over the 4-step process. It is less clear that those advantages will make a difference in practice. For example, experiments with METROPOLIS (Marchal, 2003) have shown that changing from vertical queues (with infinite vehicle storage on a link) to horizontal queues (with finite storage) leads to a lot of changes in traffic volumes at the individual link level. However, we observed that travel time structure of whole paths in the system was not very much affected as long as agents are allowed to perform within-day re-planning at each intersection. And since for many economic variables such as system performance, the travel time structure is the most important model result, this may be sufficient for many types of analysis. On the other hand, link volumes are critical to, say, emissions, and therefore in this case a more disaggregated simulation is a necessity.

This argument implies, confirmed by our experience (de Palma *et al.*, 2001), that the classical static assignment usually fails to provide both consistent travel times and volumes at the same time for congested systems. In practice, static models are calibrated against traffic counts to produce realistic volumes. However, this leads to unrealistic travel times on congested links because the volume-delay functions do not work well above the capacity threshold.

It would be good to have more systematic versions of such results, i.e. a systematic understanding of when the assignment approach yields a useful approximation of the real world and when not. Such an understanding may eventually be achieved by diligent work comparing assignment models, agent-based simulations, and measurements from reality. (An obvious way to proceed is to have benchmark cases and more thorough validation studies.) Such an approach is in some sense similar to the approach in statistical physics, where much if not all of the aggregated laws of thermodynamics or fluid-dynamics can be derived from microscopic models. That is, there is a microscopic theory, but in practice often the aggregated approaches are used. However, because of the microscopic understanding, the limitations of the aggregated approaches are well known, and it is also known which corrective terms to use in order to push the envelope of validity.

A big advantage of a good microscopic foundation of aggregate models would be that one could continue using them in cases where computational, human, or data resources are not sufficient for an agent-based model.

2.5.4 Multi-scale models

Once the capabilities and limitations of the aggregated models are better understood, one could also look into **multi-scale modeling**. This would couple high-resolution agent-based models in core areas of interest with lower-resolution aggregated models in boundary areas, significantly reducing computing times. Such approaches are pursued in many areas of science and engineering, for example in weather forecasting, where the "world model" results are used as time-dependent boundary conditions for regional models. Note, however, that a deep understanding of the multi-scale characteristics is necessary to make this a success. If, to take again an example from weather forecasting, the cold front moves faster in one model than in the other, there will be strong artifacts at the boundaries, potentially modifying the whole system dynamics. Such artifacts need to be understood and avoided before multi-scale modeling can be made a success.

2.6 Summary

This part of the paper has sketched the modeling issues that are related to multi-agent simulations. It is clear that multi-agent simulations of traveler behavior consist of many modules that need to cooperate. This text has emphasized the differences between the physical layer (the MobSim) and the decision layer (the strategy generation), as well as the importance of adaptation, learning, and feedback. Finally, it was discussed how agent-based approaches relate to the established 4-step process, implying that the transition from 4-step to agent-based can be done via incremental changes or via a drastic revolution.

Comparing agent-based to existing models, a striking difference is that in agent-based models much emphasis is put on the modularity of the overall model and on its ability to capture actual human behavior. Another important aspect is the heterogeneity of the methods that have to co-exist. We mentioned them: the MobSim layer borrows much from physics while the strategic layer borrows from a vast spectrum of disciplines: complex systems, machine learning, economics, psychology, operations research, etc. This raises several obvious questions about the computational implementation. These will be addressed in Sec. 3.

3. Computational techniques

3.1 Introduction

As mentioned at the end of the last section, an agent-based approach to travel behavior simulations leads to a large number of heterogeneous modules that need to coexist in a joint simulation environment. This raises many questions about computational implementation, such as:

- Authoring: heterogenity of models implies that different people will need to cooperate on multi-agents simulations. What are the coding tools and policies needed?
- Data management: storing the initial conditions as well as the performance of the physical layer and the agent decisions needs to be coordinated. Are the existing systems appropriate for these outputs?
- Module coupling: heterogenous modules have to agree on some protocols and some data exchange. A lot of todays' computing research is concerned about this. Which techniques are available in our case?
- Performances: realistic systems have million of agents. Even simple computations at the agent level (e.g. 10 seconds) will produce simulations that are prohibitive in term of CPU usage. Can parallelization and distributed computing help to reduce the simulation time?

3.2 Coding

Computing in the area of transportation deals with entities which are plausibly encoded as objects. Examples are travelers, traffic lights, but also links (= roads) and nodes (= intersections). This suggests the use of object-oriented programming languages, such as C++ or Java.

3.2.1 Object-oriented languages and inheritance

An important issue with the use of object-oriented programming languages is the use of inheritance. It is straightforward to start out by just having different computational classes for different classes of objects, for example for travelers, traffic lights, links, nodes, etc. However, eventually one will notice that those different classes share some similar properties. For example, they all have a unique identification tag `ID`, and it gets tiresome to reprogram the mechanics for this anew every time a new class is started. This implies that there should be a class `SimulationObject` that knows how to deal with `ID` tags. In C++, this could look as follows:[1]

```
typedef string Id ; // define that IDs are of type strings
class SimulationObject {
protected:
```

[1] We deliberately stay away from constructors here. In our experience, they are not as useful as they seem, in particular since they are not inherited, at least not in C++.

```
    Id id_ ; // ID is a string
public:
    void set_id( Id tmpId ) {
        test_if_unique( tmpId ) ;
        id_ = tmpId ;
    }
} ;
```

This could be used via

```
    ...
    // initialize object and allocate memory:
    SimulationObject* simObj = new SimulationObject() ;

    // set ID of object to ``name'':
    simObj->set_id( "name" ) ;
    ...
```

A traveler class could then inherit that functionality:

```
class Traveler : public SimulationObject {
}
```

and without any further ado one could program

```
    ...
    // initialize traveler object and allocate memory:
    Traveler* trav = new Traveler() ;

    // set ID of traveler object to ``name'':
    trav->set_id( "name" ) ;
    ...
```

This becomes of particular importance if one needs different representations of the same entities for different purposes. For example, a `link` inside a simulation needs to be able to have a traffic dynamics executed on top of it, which implies that it needs some representation of road space. In contrast, a `link` inside a shortest-path computation will not need any of this, but it may need to be aware of time-dependent link travel times. The same is true if one wants to experiment with different models: Some behavioral model of a traveler may need the age of the traveler, while some other behavioral model may be interested in driver license ownership.

To a certain extent, it is possible to live with classes that are just supersets of all these properties. For example, if a traveler class already exists and contains an `age` variable, then `driver_license_ownership` could just be added. There are, however, cases where just adding variables causes too heavy burdens on computer memory consumption, such as when two researchers want to experiment with different mental memory models inside a traveler object. In such a case, both researchers can agree to use the same base class, which provides basic functionality including, for example, reading demographic properties from file, but then both researchers inherit from this base class to have their own extensions. C++ allows multiple inheritance so that class `Traveler` can be both a `SimulationObject` and a `DatabaseObject` for example. Java allows only single inheritance.

3.2.2 Container classes

Another useful property of modern programming languages is that increasingly advanced data structures are available. The arguably most important data structures for the purpose of travel behavior simulation

are the container classes, which contain and maintain collections of objects. For example, all travelers can be stored in a single `vector` or `list` container. These container classes take care of memory allocation, that is, one does not have to know ahead of time how many elements the container will maximally contain. An important functionality of container classes are so-called iterators, which allow to go through all objects in a container in a systematic way without having to worry about array boundaries or memory faults.

In C++ , these classes are supplied by the so-called Standard Template Library (STL). They are now included by default in most modern C++ compilers and do not need any additional compiler switches. In Java, the container classes and related algorithms known as the Collections Framework have been released since the 1.2 version. The main difference between the C++ and the Java implementation of container classes is type safety: in C++ the compiler checks that objects from different types are not mixed up in a given container. In Java, object types are only checked during run time so that potential errors might still subsist in the code. The elaborated handling of exceptions in Java often allows the developer to track these errors in the early stage of development. Nevertheless, developers have since long recognized the advantage of generic programming offered by C++ templates. For this reason, a similar feature called Generics is included in the latest release (1.5) of Java (Java Generics www page, accessed 2005).

3.2.3 Garbage collection

Some languages, notably Java, offer **garbage collection**, while some other languages, notably C++, do not offer it in their standard implementation. Garbage collection refers to a process that reclaims dynamically allocated but no longer needed memory during program execution. With multi-agent code, this could for example refer to an agent's plan which is no longer needed. A sophisticated implementation would explicitely put the previously needed memory into some pool of available free memory; but a less sophisticated or faulty implementation may forget this and for example just delete the pointer (i.e. the reference) to the memory location. Garbage collection in regular intervals goes through the memory and checks for areas to which no reference exists. Such memory areas are then moved into the pool of available free memory.

In multi-agent simulation code, in contrast to the typical numerical analysis application in FORTRAN for instance, objects are constantly created and destroyed because of the dynamic nature of the underlying model. Therefore, garbage collection and the absence of pointer arithmetics reduces by a large amount the debugging process since memory allocation bugs are the most frequent coding errors. However, garbage collection is usually slow, even though it can run in a parallel thread which helps on a multi-CPU machine.

3.2.4 Swarm/RePast

Some researchers in travel behavior research use the Swarm (Swarm www page, accessed 2005) library. A similar but newer approach is RePast (Repast www page, accessed 2005). An advantage of RePast when compared to Swarm is that it is based on a wide-spread programming language (Java), in contrast to Swarm which is based on Objective C. Both approaches, and probably many others, provide library support to program systems which are composed of many agents or small software entities. The arguably most important element of support is the spatial substrate (e.g. cells, graphs, different types of boundary conditions). The functions provided include moving agents, checking for the presence of other agents, and, importantly, graphical output. In addition, the systems provides a mechanism to change parameters while the program is running (rather than having to edit some configuration file, or to re-compile the code).

Our own experience with these systems is that they are extremely useful in the prototyping of new models but potentially slow. This is certainly true when graphics are switched on, but it is often still true when they are switched off. The reasons are that RePast uses an elegant object-oriented design that sometimes makes compromises with respect to computational performances and also that Java (the basis for RePast) is usually somewhat inferior to C++ in term of computational speed. Also, the graph support inside

RePast is currently not useful for transport simulations since nodes cannot be kept at fixed geographical coordinates.

3.2.5 Readability and maintainability

An issue with object-oriented programming is code readability and maintainability. It is often claimed that object-oriented code is easier to maintain than other code. This is not consistent with our own experiences, and this statement is valid in spite of documentation support systems such as javadoc (Javadoc www page, accessed 2005) or doxygen (Doxygen www page, accessed 2005). In particular, newcomers to an object-oriented code need to possess considerable language knowledge and experience in order to do meaningful code changes. It is maybe more correct to say that object-oriented programming helps a programmer who is very fluent and experienced both in the concepts and in the technical details, and who works with a given code over a long period of time. It does, however, not necessarily help a project which is maintained by less experienced programmers, or when programmers change often, or when there is a big difference in object-oriented design preferences from one programmer to the next. It seems to us that only consistent teaching, which develops norms for "building" programs (in the same way as there are norms for planning and building bridges), will in the long run improve the situation.

3.2.6 Portability

Another issue concerns portability. At this point, the only nearly completely portable language is Java, and it is portable even with graphics. One can only hope that Microsoft will continue to support a completely standard-conforming Java.

Regarding C++, as long as code is written without any graphical items such as user interfaces or graphical display of results, it is in principle possible to keep it portable between Unix, Windows, and Mac. (Since Mac OS-X is a Unix-based system, portability of Unix-based codes to and from Mac OS-X has become rather straightforward.) Nevertheless, our experience is that porting C++ code from Linux to Windows, even when it was written with portability in mind, takes several days, and the result is not entirely satisfying, since Unix uses Makefiles while Windows uses "projects" to organize large projects.

If one wants to deliver a Windows executable only, then the Cygwin (Cygwin www page, accessed 2005) development environment may be a solution. The cygwin library comes as a Windows DLL (Dynamically Loaded Library) that wraps most of the standard Unix system calls and uses Windows system calls instead. The Cygwin development environment offers *exactly* the same tools as a Unix platform (bash, make, libc, gcc, g++, STL), but it does not provide an easy solution when someone wants to deliver a library or code that can be imported into a Windows IDE (Integrated Development Environment). Moreover, code that is dependent on the operating system layer will not be ported easily (e.g. socket I/O).

3.2.7 Computational performance of different programming languages

Common belief is that Java is considerably slower than C or C++. At least in the area of multi-agent traffic simulations, this is not consistent with our own observations. We have taught a class on multi-agent transportation simulations several times, which consisted of programming a simple traffic micro-simulation, a time-dependent fastest-path router, and a simple departure time choice model. With about half of the students programming in C++ and another half programming in Java, we rarely found important differences in terms of computational performance between the groups. The main differences that we have found are:

- Java often offers several methods to achieve similar things, for example different methods for I/O, e.g. reading and writing files. There is ample documentation in order to select which method solves which problem best, but we have found very little documentation with respect to the performance

differences of those different methods. As understandable as this is from the perspective of robustness ("leave it to the compiler" and "technologies may change anyway"), this is not acceptable when file reading times of large files can vary between minutes and hours, depending on the method selected.

- Message-passing libraries, such as MPI (see Sec. 3.6.3) are not well supported under Java: Despite some projects there is no standard, and support for special hardware such as Myrinet (see 3.6.1) is weak. This makes high performance parallel computing difficult with Java. On the other hand, there are emerging Java technologies such as Java Jini or Java RMI (Remote Method Invocation) that may be extremely appropriate for the implementation of the strategy simulation layer since they allow not only to move data accross computing nodes, but also code. However, the standard implementation of RMI suffers from large latencies (long start-up times before the first bit of a message is actually transmitted). Several projects have attempted to replace the communication layer with a high performance package to deploy RMI on clusters. See Secs. 3.6 and 3.7 for more information.
- In the past, high performance 3d graphics was not a strong point of Java. The Java3D package (Java 3D API www page, accessed 2005) is now more mature and it delegates most of the 3D intensive work to native libraries (such as Direct3D or OpenGL), that in turn delegate some of the workload to dedicated hardware. In practice, except for games and immersive simulations, Java can now be used to design medium-sized 3D applications such as used in CAD and scientific visualization.

Although the discussion was based on these two languages, it should be noted that there is also growing interest and enthusiasm for Python, which is an object-oriented scripting language. For instance, URBANSIM (and RePast as well) is currently being ported from Java to Python, and there is an emerging international collaboration towards a computational framework for land-use simulations, possibly also written in Python (Waddell *et al.*, 2005). The reasons behind this move are the simplicity of the code and the ease of use, from inside Python, of efficient C libraries that are geared toward matrix computation. In our opinion, this experience is not transferable, as a general rule, to simulations that deal with a lot of agent-to-agent interactions (such as the MobSim). Indeed, the performance limiting factor in those simulations is the speed at which objects can be randomly accessed from the memory core. This explains why C++ and Java lead to similar performances: the current CPUs are extremely fast compared to available memory technologies. Nevertheless, a major advantage of Python is its readibility and, therefore, its appeal to modellers with less training in computer science.

3.3 Search methods

All the strategic parts of a travel behavior simulation are concerned with search: Finding links that combine to a good route; finding activity locations that suit individual preferences and constraints; finding good activity times; finding a residential location; etc. And many areas of computer science are indeed concerned with search. This reaches from purely engineering issues ("how to find a file on a hard disk") via operations research (e.g. traveling salesman problem) to cognitive modeling and Artificial Intelligence.

For travel behavior research, it is important to know these methods, since many models in travel behavior research essentially lead to search problems that cannot be solved within reasonable time on existing or future computers. Take nested logit models, which can be visualized as a decision tree. For example, one can phrase activity-based demand generation as a nested problem, consisting of pattern choice, location choice, time choice, and mode choice (e.g. Bowman, 1998). An agent faced with the activity scheduling task first makes the pattern decision, then the location decision, etc. The decision at each level depends on the expected utility of each sub-tree. This means that, in order to compute the decision, one needs to start at the bottom leaves and iteratively propagate the utilities to the upper branches.

To understand this, let us index patterns with p, locations with l, times with t, and mode with m. The utility of a complete decision is denoted U_{pltm}. Now, for a given plt, the algorithm evaluates the different utilities for all m, and from that computes the expected utility for the choice plt. And here lies an important difference between different choice models: While in most computer science search algorithms, this

would just be the maximum utility (i.e. $U_{plt} = \max_m U_{pltm}$), in random utility theory it is the *expected* maximum utility. For nested logit models, this is the famous logsum term $U_{plt} \propto \ln \sum_m e^{U_{pltm}}$.

The algorithm then computes utilities for all t given pl, and from that computes the expected utility for the choice pl. This continues until the computation has reached the root of the decision tree. At this point, the algorithm turns around and goes down the decision tree, at each decision point making the best choice. Again, in computer science search, this choice would just be the best option, while in nested logit, an option is chosen with a probability proportional to $e^{\beta U_i}$.

This seemingly small difference between "selecting the max at each decision point" and "selecting according to $e^{\beta U_i}$ at each decision point" has very important consequences in terms of the search complexity: While the choice of the maximum allows to cut off branches of the search tree that offer no chance of success, in probabilistic choice *all* branches of the search tree need to be evaluated. This means that many efficient computer science methods cannot be used for random utility theory. More distinctly, *evaluating a model using random utility theory is much more costly than evaluating a model using plain maximization*. Now since many of the behavioral search problems cannot even be solved optimally using "max" evaluation, there is little hope that we will be able to directly solve the same problems using the even more costly random utility framework.

In consequence, at this point it makes sense to look at alternatives, such as the following:

- Sometimes, it is possible to use max-based search algorithms to compute not just the best option, but also second-best, third-best, etc. This is for example well known for shortest path computations (e.g. Perko, 1986). That is, one could limit the choice set to some k-best options.[2] In many cases, however, computing a second-best option is much more expensive than computing the best option.
- Random utility is in effect a noise parametrization: Agents do not compute the utilities that the scientists compute, but modify them by some random disturbances. However, stochastic simulations change from one iteration to the next anyway; that is, computing the fastest path based on each of the iterations of a Dynamic Traffic Assignment will generate a choice set, with different frequencies for each option. It would be interesting if this feature could be exploited, rather than trying to fight it as what is currently done by first averaging over many iterations to obtain a deterministic result and then use random utility modeling to re-introduce the noise.
- Computer science knows many search heuristics. These heuristics come up with useful solutions within limited resources; sometimes, they come together with some quality guarantee (such as "not more than 10% worse than the optimal solution, even if we cannot compute the optimal solution"). Now, from a behavioral point of view, it seems that when a computer cannot reliably find an optimal solution, it is improbable that humans can do so. In consequence, it would make sense to look at computer science heuristics, and to compare them to heuristics based on human psychology.

  In this context, note that route choice is maybe *not* a good field to explore this since route choice is one of the few fields where the optimal solution, albeit maybe not behaviorally realistic, is extremely cheap to compute, much cheaper than most heuristic solutions.

A relatively new set of heuristic search algorithms are "evolutionary algorithms". This is a field that looks at biology and evolution for inspiration for search and optimization algorithms. Examples are Genetic Algorithms (Goldberg, 1989), Neural Nets, Simulated Annealing, or Ant Colony Optimization (Dorigo *et al.*, 1999). These algorithms compete with methods from Numerical Analysis (such as Conjugate Gradient) and from Combinatorial Optimization (such as Branch-and-Bound). In our experience, the outcome in these competitions is consistently that, *if* a problem is well enough understood that methods from Numerical Analysis or from Combinatorial Optimization can be used, then those methods are orders of magnitude better. For example: There is a library of test instances of the traveling salesman problem (TSPLIB www page, accessed 2003), and there is a collection of the computational performances of many different algorithms when applied to those instances (TSP challenge www page, accessed 2003). The largest instances of the traveling salesman problem that a typical evolutionary algorithm solves, using

[2] Note that k-shortest paths normally have a lot of overlap, and need additional measures within the random choice framework (e.g. Cascetta *et al.*, 1996).

several hours of computing time, are often typically solved by a Combinatorial Optimization algorithm in much less than a second.

In contrast, evolutionary algorithms are always useful when the problem is not well understood, or when one expects the problem specification to change a lot. For example, an evolutionary algorithm will not complain if one introduces shop closing times into an activity scheduling algorithm, or replaces a quadratic by a piecewise linear utility function (Charypar and Nagel, 2005). Faced with such problems, the evolutionary algorithms usually deliver "good" solutions within acceptable computing times and with relatively little coding effort.

This characterization makes, in our view, evolutionary algorithms very interesting for travel behavior research, since both according to random utility theory and according to our own intuition, this is exactly what humans do: find good solutions for problems within reasonable amount of time, and consistently find such good solutions even when circumstances change.

An interesting research issue would be to explore the distribution of solutions in the solution space. Maybe there are evolutionary algorithms that generate solutions of quality U_i with probability e^{U_i}, but in much less time than the full search algorithm.

3.4 Databases for initial/boundary conditions

The necessity of initial and boundary conditions was described in Sec. 2.1. There is probably very little debate that databases are extremely useful to maintain information about road networks or census data. There are now many commercial or public domain products which support these kinds of applications. Standard databases are always easy to use when the data consist of a large number of items that all have the same attributes. For example, a road link can have a from-node, a to-node, a length, a speed limit, etc.

The two most common situations are that of a GIS database and that of a database management system (DBMS) using an SQL relational database. Note that most GIS vendors provide gateways to link their software to SQL DBMS. Besides data management issues, the main advantages of using SQL DMBS is their flexibility. Data can exported and imported between simulation modules that are totally independent and useful statistics can be extracted easily. URBANSIM (Waddell *et al.*, 2003) and METROPOLIS (de Palma and Marchal, 2002) use that approach to store input and output data. The performance bottleneck of SQL DBMS is the sorting of records and the insertion of new records. When these individual operations are performed at a very high rate, such as inside a simulation loop, the usage of a SQL DBMS severly deteriorates the overall performance. But when these operations are performed in bulk, these systems can be very efficient. For instance, at the end of a METROPOLIS simulation, the insertion of 1 million agents with 15 characteristics each in the underlying MySQL database takes only 2 sec. on a Pentium 4. MySQL (MYSQL www page, accessed 2005) is fast becoming a de facto standard because it is open-source, easy to manage, performant and it already integrates some geographic features of the OpenGIS specification (OpenGIS www page, accessed 2005).

However, even the data for initial and boundary conditions is not entirely static. Changes can happen for the following reasons:

- **Keeping it current:** As time moves on, new streets are built and older ones are changed to pedestrian zones. The database needs to keep track of these changes.
- **Errors:** Any data contains errors, and all data *including the historical data* should be continuously improved over time.
- **Higher detail:** Data can be enhanced or replaced by data of higher detail. This could for example mean the inclusion of low capacity roads, the addition of public transit, higher resolution for the census data, or any other information.
- **Scenarios:** Most simulation applications in traffic refer to case studies, where some elements of the scenario are changed while all others are kept fixed. For example, a street could be added to or removed from the road network data, while all other data remain fixed.

3.4.1 Differences to "standard" database applications

The arguably main commercial application of standard databases are financial transactions. The task of such a system is to maintain a consistent view of the whole system at each point in time, to do this in a very robust way that survives failures, and to maintain a record of those states and the transactions connecting them.

Database applications for initial and boundary conditions are, in contrast, rather different. Our data is not defined by the transactions; rather, transactions are used to define the data. That is, a standard database will solve the problem of keeping the data current, but it will not solve the problem of retroactive changes, which are imposed by error corrections, higher detail imports, or scenarios. There seems to be very little database support for such operations. Some examples:

- Just replacing supposedly erronous data entries by supposedly more correct ones is not a viable option since in the long run nobody will know any more to what status the data corresponds. *Any* change in an existing data set needs to be documented; as a minimum requirement, there should be a complete trail of all changes together with the date of change and the name of the person authorizing the change.
- Changing data entries during the course of a case study is not possible, because simulation runs would no longer be comparable. This refers even to the correction of (supposedly) erroneous data items: If a study was started with certain errors in a data set, it will have to be finished with those same errors.

  This makes it impossible to keep a database up to date while some other team is using it to run a study. The record and production of snapshots of the database is thus a crucial feature.

Diligence and discipline by the operators can overcome some of these issues. Records can contain fields which denote their temporal range of validity. For example, if a road link gets converted into a pedestrian link at the end of the year 2000, the whole link record in the database could be copied, the copy could be modified to reflect the new status, and then the first record could be marked as being valid until the year 2000, while the second record could be marked as being valid starting in 2001. Scenarios could be treated in a similar way, that is, the database contains a superset of all information ever used in the whole study, and flags denote which information is switched on or not for each specific case. The German company PTV has used a similar technique for the computation of all scenarios for the so-called Bundesverkehrswegeplan (Vortisch, personal communication).

When thinking about this, one recognizes that a similar mechanism needs to be employed for error correction. Essentially, it is necessary that data entries are *never* deleted. If one wants to correct supposedly erroneous data entries, then the whole record needs to be copied, then the erronous element is corrected in the copy and eventually flags are adjusted so that the situation is clear. In that way, a study relying on the faulty record can still use that faulty record until the whole study is finished.

3.4.2 Version control systems

Now while it is clear that the above is feasible, it is also clear that it will be prone to errors. In particular, there is the all-too-human tendency to correct errors without telling anybody. In consequence, some software support for these operations would be useful. With current databases, it is possible to maintain a log of all operations, and so it is in principle possible to go back and find out who changed which entry when. This is, however, not sufficient: What is needed is a system that allows us to retrieve some data in the status it was at a certain point in time, no matter what the later changes are. And in fact, there are systems which do exactly what we want, but unfortunately only for text files and not for databases. These systems are called **version control systems**, and they are for example used for software development. The "change log" system of some text processors such as MS-Word and Openoffice is a related, albeit simpler system. An example of a version control system is CVS (CVS www page, accessed 2005); it has the advantage that it works across different operating systems and that it allows remote access. Nearly all public domain software these days uses CVS.

Version control systems provide the following functionality:

- There is a main trunk of development. All changes that anybody ever does to the main trunk are recorded, together with a date, user name, and an optional comment.
- There is the option of branches. The typical use for branches are patches for stable software releases. Say that a software is released and used by others, while the main trunk goes into an experimental development phase. If at this point bugs are found in the released version, it will not be possible to fix those bugs in the experimental version and release that to users. Instead, one will make a branch which starts at the released version, and apply patches to that version.
- Last, there is the option of merges. For example, in the above example it is possible to merge the patches into the experimental version.
- CVS also offers *concurrent* development, that is, more than one developer can work on the same files. For a variety of reasons, this is probably not relevant for the database issues discussed here.

The main point of shortly digressing into software version control systems is to show that systems which have the desired functionality are in existence and have been used for many years. This implies that it is a technology which is well understood and more than ready for release. The main reason why there seems no widespread use of version control technology in the database area is that the main customer of the database industry is the financial services industry – and that industry is mostly interested in a consistent view of the *current* situation, and not in the running of different scenarios. It is sincerely hoped that this situation will change in the future, and that some databases will be geared more towards scientific applications. Since other scientific areas, such as bio-informatics or high energy physics, are now moving into the wide-spread use of large scale databases, there is hope that this will happen soon. If it does not happen, then the transportation community should get together and design at least some minimal version, which would essentially be an "add-only" database, where entries, once entered, can never be removed. They could only be invalidated by a flag.

3.4.3 What to store/data aggregation

In our view, it makes sense that the primary data for initial and boundary conditions is as close to the physical reality as possible, since in our view that is the best way to obtain an unambiguous description. For example, it makes more sense to store "number of lanes", "speed limit", and "signal phases", rather than "capacity", since the latter can be derived from the former (at least in principle), but not the other way round. In part, this will be imposed onto the community by the commercial vendors, which have an interest to sell data that is universally applicable, rather than geared to a specific community.

If the primary source of data is highly disaggregated, it is tempting to keep this high level of detail all through the forecast models, and projects such as TRANSIMS, METROPOLIS, and MATSIM develop large-scale simulation models that may, in the long run, be able to cope with highly disaggregated data for the physical layer and thus avoid any aggregation. However, this is not always feasible from the computational point of view, and therefore systems that run on more aggregated data will continue to exist. Unfortunately, the aggregation of much of traffic-related data is an unsolved problem – while it is straightforward to aggregate the number of items in cells in a GIS to larger cells, aggregating network capacity to a reduced graph is far from solved. There needs to be more research into multi-scale approaches to travel behavior simulations.

Even if these problems get eventually solved, we firmly believe that models should make the aggregation step transparent. That is, there should be universally available data, which, for the ambiguity reasons stated above, should be as disaggregated as possible, and then aggregation should appear as an explicit feature of the model rather than as a feature of the input data. Conversely, the results/forecasts should be, if possible, projected back to the original input data.

3.4.4 Data merging

Another issue with respect to databases that probably many of us have faced is the merging of data sets. Let us assume the typical case where we have some large scale traffic network data given, and some sensor locations registered for that network. Now we obtain a higher resolution description of a smaller geographical area. One would like to do the following:

- Overlay the second data set on the first one. Remove the items from the original data set that are superceeded by the higher resolution set and stitch the road networks together.
- Move the registration of the sensors, where applicable, to the new data set.

It would be possible to write heuristics which could solve most of the problem semi-automatically, i.e. in a way where uncritical cases are resolved automatically, and problematic situations are automatically presented to an analyst. This would be similar to the automatic proof checking technology applied in some software projects. A first implementation of this is now available (Balmer *et al.*, 2005).

3.4.5 The possible role of commercial data vendors

Some of the problems of data consistency will go away with the emergence of a few internationally operating data vendors. They will define the data items that are available (which may be different from the ones that we need), and they will make consistent data available at pre-defined levels. For example, it will be possible to obtain a data set which contains all European long-distance roads, and we would expect that it will be possible to obtain a consistent upgrade of the Switzerland area to a higher level of detail.

We are a more sceptic towards the commitment of such companies to historical data. With little cost, companies could make snapshots of the "current view of the current system", and one could thus obtain something like the "year 2000 view of the year 2000 system". There would, however, also be the "current view of the year 2000 system", or "intermediate views of the year 2000 system". The first would be useful to start a historical study; the second would be useful to add data consistently at some later point in time.

Finally, there will probably always be local error correction of the type "company XY always gets A and B wrong and so we always correct it manually". Obviously, such changes need to be well-documented and well-publicized.

3.4.6 Scenario maintenance

In our view, there is no system that manages the scenario data changes consistently as outlined above; in consequence, there is also no simulation system that manages consistently dependencies of simulation result on input data. A system should at least be able to remember what has happened to the data and what part of the forecast it affects. Some transportation planning software systems have made progress in that direction, e.g. EMME/2 and METROPOLIS.

EMME/2 (INRO Consultants Inc., 1998) comes with a binary-formatted, proprietay database (the so-called databank file). This file contains the description of the transportation network (links with lanes, length, capacities) and the travel demand in the form of OD matrices. EMME/2 proposes the definition of *scenarios* that consist in pairing networks and matrices and performing the assignment. EMME/2 designers recognized earlier the importance of tracing back all the modification done to the complex network layer: each change of a single attribute of the links is recorded and can be brought back. This feature is reminiscent of the *roll-back* feature of relational databases. However, once an assignment is performed, the output variables (here the traffic volumes) do not follow the same handling: these variables are erased after each assignment and have to be kept manually in user-defined vectors. The modifications of the initial data does not invalidate the results. However, EMME/2 provides the feature of multiple scenarios so that the end-user can modify the network of another scenario, run the assignment, and compare the scenarios. It is left to the discipline of the user not to modify the initial data thereafter.

METROPOLIS uses the same network description (not storage) as EMME/2. It uses also an OD matrix but with an extra layer of agent description called "user types" that define user constraints in terms of schedule. METROPOLIS uses an open SQL relational database (MySQL) to store all the initial data it needs: networks, matrices, users, scenarios. The roll-back feature is delegated to the database. The idea is that, even if it useful to be able to roll-back, it is often unusable when the number of components that have changed is huge. For instance, if all the capacities of the network are multiplied by 1.1, it will not be possible to understand the modifications unless it is documented with a log. However, METROPOLIS maintains the consistency of data throughout the simulation process: no data can be modified that have been used in a completed simulation. METROPOLIS distinguishes different bulks of data sets: networks, matrices, user-types, function sets, demands, supplies and finally, simulations (Fig 4). Each of these major blocks encapsulates a whole set of disaggregated data (e.g. networks have thousands of links). This decomposition is arbitrary and only intented to ease the use of creating scenarios by mixing variants. Each of these data sets can be in one of the 3 different states: free, editable or locked. In the free state, the data block can be erased/modified without compromising results or other data sets. In the editable state, other data blocks depend on this one and so this one cannot be deleted without deletion of the dependent blocks; however, no simulation has been performed using it and it can still modify its internal attributes. In the locked state, it means that some simulation has been or is using the data set and so the data set cannot be modified. By doing so, the METROPOLIS inner database ensures that whatever result is still in the database contains also the exact initial data set of the simulation. However, no log is kept of the modifications that might have been done to generate the initial set and it is left to the end-user to write something in the *comment* attribute of each data block. Each data block can be duplicated/deleted/documented to ease the replication of experiments and the comparisons between forecasts that share some data blocks.

3.5 Module coupling

3.5.1 Introduction

As pointed out earlier, multi-agent transportation simulation systems consist of many cooperating modules, such as the synthetic population generation module, the housing market module, the activity generation module, etc. An important issue for the future development of multi-agent transportation simulations is the coupling between those modules, i.e. how they cooperate to form a coupled simulation system.

Despite many efforts, the easiest solution in terms of the implementation of a simulation package is still to have a single person write a complete system for a single CPU on a single operating system (OS). In addition, all tightly coupled modules should use the same programming language since that considerably simplifies the coding of module interaction.

With multi-person projects, the argument remains the same: The easiest approach is to program for a single-CPU system and have everybody use the same OS and the same programming language at least for tightly coupled modules. However, as models evolve, it is desireable to be able to plug-in new modules without compromising the whole architecture. This section will explore technologies for this purpose.

The coupling of the modules also depends on their synchronization. Obviously, the organization of how user decisions take place on the time scale affects considerably the way the modules will interact. When decisions are revised frequently in the simulation, it imposes to have a coupling implementation that is computationally efficient.

3.5.2 Module coupling with subroutine calls

There are many ways to couple modules and all methods have their own advantages and disadvantages. The first one that comes to mind is to couple them via **regular subroutine calls**. For instance, the MobSim can be suspended at regular intervals and each agent is asked if he/she wants to replan. If the answer is yes, the corresponding subroutine is called. Once all agents are treated, the system returns to

the traffic simulation thread.

The advantages of this method are that it corresponds to long-established programming methods, and that its logical scheduling is easy, in the sense that while a traveler "thinks", the flow of the real world is suspended. This makes it rather easy to model all kind of thought models, including immediate or delayed response.

The disadvantages of this method are that the different methods need to be compatible on some level, which usually means that they are written in the same programming language on the same OS. In addition, the approach is not amenable to a parallel implementation, as will be discussed in Sec. 3.7.2. The more strategic modules one includes, the less time the CPU will have to advance the simulation of the physical system. Large scale scenarios are infeasible with this method.

3.5.3 Module coupling with files (including XML)

Often, different strategy generation modules come from different model writers. Some group might provide a router, some other group an activity generation module, and the next group a housing choice module. All of these modules may be coded in different programming languages and on different OSs. For such modules, coupling via subroutine calls is no longer possible. In such a case, the maybe first thing that comes to mind is to couple them via files. This approach is in fact followed by many groups, including TRANSIMS and some of our own work.

A question here is which file format to use. TRANSIMS uses line-oriented text files. This works well as long as the data are column-oriented, such as files for links and nodes. It becomes considerably more awkward with respect to agent information, which is less well structured:

- Strategies of agents may contain variable-length pieces, such as a route. Also, there may be hierarchical information, such as demographic information (on the agent level), information about different alternatives, or information about every specific decision point.
- Not all agents may have the same information. This makes the column-oriented approach difficult, since for each additional information item a new column needs to be started, even if it remains unused by most agents.

A plausible way out of the dilemma is offered by the XML format (XML www page, accessed 2005). Rather then dwelling on general descriptions, let us look at a possible example:

```
<person id="13" income="50kEuro/yr" >
    <plan score="561">
        <act type="h" location="..." end_time="06:00" />
        <leg num="0" mode="car" expected_trav_time="00:15:04">
                <route>2 7 12</route>
        </leg>
        <act type="w" location="..." dur="08:00" />
        <leg num="1" mode="car" expected_trav_time="00:39:04">
                <route>13 14 15 1</route>
        </leg>
        <act type="h" location="..." link="1" />
    </plan>
    <plan score="463">
        ...
    </plan>
</person>
```

The example denotes a traveler agent with two plans, one with score ($\approx$ utility) of 561 and one with score 463. The first plan is given in more detail, it consists of an activity chain of "home–work–home",

together with some location information whose exact type (e.g. street address, x/y coordinate, link ID) is not specified here. In addition, there is information on activity scheduling times, and on travel times.

The point here is not to debate the details of the data entries; the two important points are the following:

- **Consolidated person information.** All information about a person is at one place. In fact, in our own current work we just use one single file format to exchange agent information between all modules. Higher level modules, such as activity planning, may not need lower level information such as route information and thus they may ignore the corresponding fields or leave them empty. However, with this file format, all modules have always access to all information concerning an agent. For example, the routing module has access to the demographic data. This is in stark contrast to some earlier approaches, notably TRANSIMS, and it is in our view much more amenable to the agent-based view. It is also much more amenable to true behavior-based modeling.
- **Extensibility.** The "X" in XML stands for extensible. It is possible to add information into the above file without breaking existing parsers. This means that anybody can experiment with additional items to the format without having to adapt all other modules. It is correct that the same thing can be achieved, within limits, with a column-based format, but XML is certainly much more flexible and powerful in what it can do.

Somewhat contrary to expectations, file size with XML is not a major issue. Our own experience is that uncompressed XML files are larger than the corresponding column-oriented TRANSIMS files, but once compressed, the XML files are considerably smaller. We attribute this to the fact that repeated tags such as `<person>` are easy to compress, and to the fact that column-oriented files have the tendency to contain a lot of unnecessary "zeroes" since many of the columns may not be used by most of the agents.

The arguably easiest way to implement file-based module coupling is to do it together with period-to-period (e.g. day-to-day, year-to-year) replanning: The simulation of the physical system is run based on precomputed plans and the behavior of the simulation is recorded, some or all of the strategic modules are called, the simulation is run again, etc.

The file-based approach solves possible data exchange issues between different programming languages and different OS, but it does not solve the performance issue: The assumption is still that at any given point in time, only a single CPU is advancing the simulation. It is, however, possible to parallelize each individual module. This will be discussed in Sec. 3.6.

3.5.4 Module coupling with remote procedure calls (e.g. CORBA, Java RMI)

An alternative to files is to use remote procedure calls (RPC). Such systems, of which CORBA (Common Object Request Broker Architecture; CORBA www page, accessed 2005) is an example, allow to call subroutines on a remote machine (called "server") in a similar way as if they were on the local machine (called "client"). There are at least two different ways how this could be used for our purposes:

1. The file-based data exchange could be replaced by using remote procedure calls. Here, all the information would be stored in some large data structures, which would be passed as arguments in the call.
2. One could return to the "subroutine" approach discussed in Sec. 3.5.2, except that the strategic modules could now sit on remote machines, which means that they could be programmed in a different programming language under a different OS.

Another option is to use Java RMI (RMI www page, accessed 2005) which allows Remote Method Invocation (i.e. RPC on java objects) in an extended way. Client and server can exchange not only data *but also pieces of code*. For instance, a computing node could be managing the agent database and request from a specific server the code of the module to compute the mode choice of its agents. It is easier with Java RMI than with CORBA to have all nodes acting as servers and clients and to reduce communication bottlenecks. However, the choice of the programming language is restricted to Java.

It is important to notice the difference between RPC and parallel computing, discussed in Sec. 3.6. RPCs are just a replacement for standard subroutine calls which are useful for the case that two programs that need to be coupled use different platforms and/or (in the case of CORBA) different programming languages. That is, the simulation would execute on different platforms, but it would not gain any computational speed by doing that since there would always be just one computer doing work. In contrast, parallel computing splits up a single module on many CPUs so that it runs faster. Finally, distributed computing (Sec. 3.7) attempts to combine the interoperatility aspects of remote procedure calls with the performance aspects of parallel computing.

The main advantage of using of CORBA and other object broker mechanisms is to glue heterogeneous components. Both DynaMIT and DYNASMART projects use CORBA to federate the different modules of their respective real-time traffic prediction system. The operational constraint is that the different modules are written in different languages on different platforms, sometimes from different projects. For instance, the graphical viewers are typically run on Windows PCs while the simulation modules and the database often are carried out by Unix boxes. Also, legacy software for the data collection and ITS devices need to be able to communicate with the real-time architecture of the system. Using CORBA provides a tighter coupling than the file-based approach and a cleaner solution for remote calls. Its client-server approach is also useful for critical applications where components may crash or fail to answer requests. However, the application design is more or less centered around the objects that will be shared by the broker. Therefore, it looses some evolvability compared to XML exchanges for instances.

3.5.5 Module coupling with WWW protocols

Many people know from personal experience that it is possible to embed requests and answers into HTTP protocols. A more flexible extension of this would once more use XML. The difference to the RPC approach of the previous section is that for the RPC approach there needs to be some agreement between the modules in terms of objects and classes. For example, there needs to be a structurally similar "traveler" class in order to keep the RPC simple. If the two modules do not have common object structures, then one of the two codes needs to add some of the other code's object structures, and copy the relevant information into that new structure before sending out the information. This is no longer necessary when protocols are entirely based on text (including XML); then there needs to be only an agreement of how to convert object information into an XML structure. The XML approach is considerably more flexible; in particular, it can survive unilateral changes in the format. The downside is that such formats are considerably slower because parsing the text file and converting it into object information takes time (e.g. Cetin, 2005).

3.5.6 Module coupling via databases

Another alternative is to couple the modules via a database. This could be a standard relational database, such as Oracle (ORACLE www page, accessed 2005) or MySQL (MYSQL www page, accessed 2005). Modules could communicate with the database directly, or via files.

The database would have a similar role as the XML files mentioned above. However, since the database serves the role of a central repository, not all agent information needs to be sent around every time. In fact, each module can actively request just the information that is needed, and (for example) only deposit the information that is changed or added.

This sounds like the perfect technology for multi-agent simulations. What are the drawbacks? In our experience, the main drawback is that such a database is a serious performance bottleneck for large scale applications with several millions of agents. Our own experience refers to a scenario where we simulated about 1 mio Swiss travelers during the morning rush hour (Raney and Nagel, 2003). The main performance bottleneck occurred when agents had to choose between already existing plans according to the score of these plans. The problem is that the different plans which refer to the same agent are not stored at the same place inside the database: Plans are just added at the end of the database in the sequence they are generated. In consequence, some sorting was necessary to move all plans scores of a given agent together into one location. It turned out that it was faster to first dump the information triple

(travelerID, planID, planScore) to a file and then sort the file with the Unix "sort" command rather than first doing the sorting (or indexing) in the database and then outputting the sorted result. All in all, on our scenario the database operations together consumed about 30 min of computing time per iteration, compared to less than 15 min for the MobSim. That seemed unacceptable to us, in particular since we want to be able to do scenarios that are about a factor of 10 larger (24 hours with 15 mio inhabitants instead of 5 hours with 7.5 mio inhabitants).

An alternative is to implement the database entirely in memory, so that it never commits to disk during the simulation. This could be achieved by tuning the parameters of the standard database, or by re-writing the database functionality in software. The advantage of the latter is that one can use an object-oriented approach, while using an object-oriented database directly is probably too slow.

The approach of a self-implemented "database in software" is indeed used by URBANSIM (Waddell *et al.*, 2003; URBANSIM www page, accessed 2003). In URBANSIM there is a central object broker/-store which resides in memory and which is the single interlocutor of all the modules. Modules can be made remote, but since URBANSIM calls modules sequentially, this offers no performance gain, and since the system is written in Java, it also offers no portability gain. The design of URBANSIM forces the module writers to use a certain canvas to generate their modules. This guarantees that their module will work with the overall simulation.

The Object Broker in URBANSIM originally used Java objects, but that turned out to be too slow. The current implementation of the Object Broker uses an efficient array storing of objects so as to minimize memory footprint. URBANSIM authors have been able to simulate systems with about 1.5 million objects (Salt Lake city area).

In our own work, we use an object-oriented design of a similar but simpler system, which maintains strategy information on several millions of agents. In that system, 1 million agents with about 6 plans each need about 1 GByte of memory, thus getting close to the 4 GByte limit imposed by 32 bit memory architectures.

Regarding the timing (period-to-period vs. within-period replanning), the database approach is in principle open to any approach, since modules could run simultaneously and operate on agent attributes quasi-simultaneously. In practice, URBANSIM schedules the modules sequentially, as in the file-based approach. The probable reason for this restriction is that there are numerous challenges with simultaneously running modules. This will be discussed in Sec. 3.7.

3.6 Parallel computing

In this paper, we will distinguish between parallel and distributed computing. By parallel computing we mean that an originally sequential module uses a parallel computer to break it down into sufficiently small problems that can fit into the memory of a single system or to improve the execution speed of the module. While the memory issue will become less important with the advent of 64-bit PC CPUs that can address more than 4 GByte of memory, the performance issues will not go away. It is an obvious incentive to distribute agents on different computers and to apply the same code in parallel to each subset. But different agents may need different decision modules at the same simulation epoch. For instance, once an agent selects his/her travel mode, a transit line choice module is invoked or a car route module is invoked, which may be totally different. Therefore, the distribution of agents on different nodes implies the cooperation of heterogeneous modules, which will be addressed in the section about distributed computing. This section focuses strictly on the first aspect, i.e. the homogeneous acceleration of a single module by the use of more than one CPU.

3.6.1 Hardware

Let us first look at a parallel computer from a technical perspective. There are currently two important concepts to parallel computing, shared memory and distributed memory.

Distributed memory machines are easy to explain: They behave like a large number of Pentium computers coupled via a fast network. Each CPU obtains a part of the problem, and the different processes on the different CPUs communicate via messages. One can maybe differentiate three different classes of distributed memory computers:

- So-called **Beowulf clusters** use standard desktop computers (such as Pentium Boxes) coupled by standard local area network technology (such as Ethernet). They are often called **Linux clusters**, because they often run Linux as Operating System.

  The advantage of Beowulf clusters is that, because they use commodity hardware, their cost per CPU cycle is relatively low.

- Sometimes, a Beowulf cluster is sufficient for a certain purpose except that communication is too slow. In such cases, there is special communication hardware for such clusters such as Myrinet or SCI (Scalable Connect Interface) (e.g. Kurman and Stricker, 1999). This approximately doubles the cost of each computational node.

- Finally, there are distributed memory supercomputers such as the Cray T3E. However, most current parallel supercomputers such as the SGI Origin or the IBM Regatta use multi-CPU shared memory architectures (see below) as computational nodes, which are then connected via a high-speed communication network. Thus, these are in effect hybrid architectures which behave like shared memory architectures up to a certain number of CPUs, and like distributed memory machines beyond.

In contrast to distributed memory machines, in **shared memory** machines all CPUs are combined in a single computer. Shared memory machines can be understood as an extension of the dual-CPU Pentium box, that is, many processors (currently up to 128) share the same memory. Beyond about four CPUs, the necessary hardware becomes rather expensive, since it is necessary to give each CPU full bandwidth access to every part of the large memory. Additional overhead is incurred because the shared memory needs to be locked and unlocked when one of the CPUs wants to write to it in order to avoid corrupted data.

The result is that distributed memory machines can be a factor of ten cheaper than shared memory machines for the same number of CPU cycles. However, shared memory machines are sometimes easier to program, and they are faster when a program uses many non-local operations (operations where each CPU needs information from most if not all other CPUs). They also usually have more reliable hardware than the low-cost Beowulf clusters. Nevertheless, in transportation large scale shared memory machines are rarely used, because of two reasons:

- Academic departments prefer distributed over shared memory machines, because with distributed memory machines they get more computing power for the same money, and the higher cost of programming can be absorbed via the relatively cheap student labor force.

- With a 32-bit architecture such as Pentium, one can at most address 2^{32} Byte $= 4$ GByte of memory. In consequence, if several CPUs in a 32-bit shared memory machine want to share more than 4 GByte of memory, complicated memory mapping operations need to be performed. This problem naturally limits the number of CPUs in 32-bit shared memory machines; the practical limit seems to be at eight CPUs.

  While UNIX systems from SGI, HP or SUN have had 64-bit architectures for many years now, they have been introduced only recently for PC systems based on Intel or AMD CPUs.

For distributed memory machines, information needs to be exchanged between the computational nodes. This is typically done via **messages**. Messages are easiest implemented via some kind of message passing library, such as MPI (MPI www page, accessed 2005) or PVM (PVM www page, accessed 2005). Besides initialization and shutdown commands, the most important commands of message passing libraries are the send and receive commands. These commands send and receive data items, somewhat comparable to sending and receiving postal mail.

In contrast, jobs on shared memory machines can exchange information between jobs by simply looking at the same memory location. Such jobs that share the same memory area are often called **threads** (although the word "threads" is also used in the more general context of "parallel execution threads").

3.6.2 Different degrees of parallelism

Within the area of parallel computing, there are different degrees of parallelism that can be exploited:

- Possibly, one has completely independent execution threads. This is for example the case for non-interacting agents that compute their next-day strategies. In such a case, it is possible to start all threads simultaneously on as many CPUs as are available and just wait until all of them are finished.

  The speed-up in this case is usually linear, that is, p CPUs lead to a factor of p increase in execution speed.

- Another case is that one has dependent threads, but a good way to decompose them. This is often true for spatially-oriented simulations, where one can partition space (**domain decomposition**, see Fig. 5).

  The prime example for this case, with linear speed-up but tight coupling, are all time-stepped simulations where the update of each simulation entity (objects, cells, ...) from time t to time $t + h$ depends on local information from time t only. In that case, one can update all objects simultaneously, followed by an information exchange step.

  The speed-up in this case is usually linear, but often advanced communication hardware is necessary for this (Cetin and Nagel, 2003).

- Finally, there are programs that are difficult or impossible to parallelize. An intuitive example is the addition of $2p$ numbers on p CPUs. This can be done in logarithmic time (i.e. $\propto \log_2 p$): In the first iteration, the two numbers on each CPU are added. Then one half of the CPUs sends their numbers to the other half of the CPUs. The receiving CPUs add up the two numbers they now have. This continues until there is eventually one CPU which has the result.

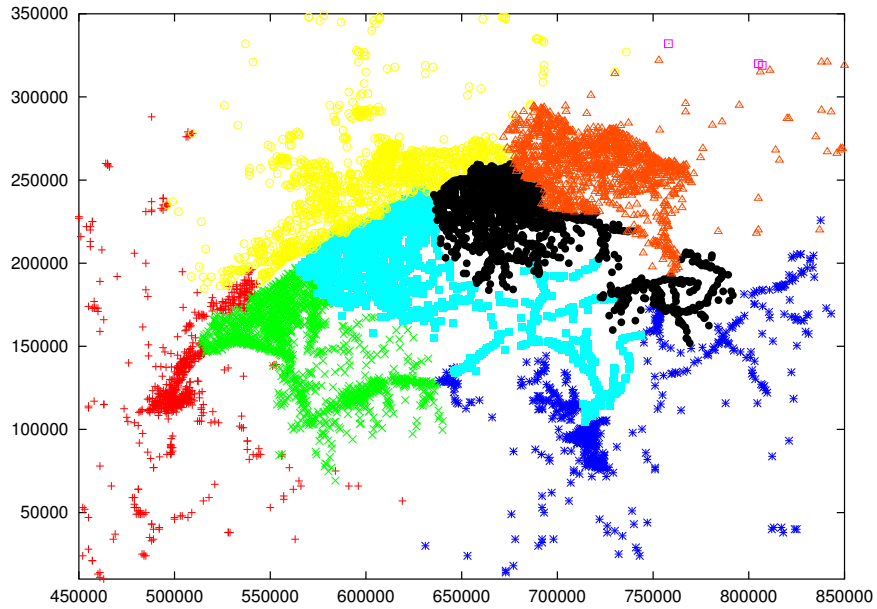
  Another typical example for this case are discrete event simulations, where each event potentially depends on the previous event. If they underlying system is a physical system, then there are limits on the speed of causality that one can exploit. For example, it is clearly implausible that a speed change of a vehicle influences, within the same second, another vehicle that is several kilometers away. There is some mathematical research into such dependency graphs (e.g. Barrett and Reidys, 1997). Also note results for parallel dynamic shortest path algorithms (Chabini, 1998).

3.6.3 Software for parallel computing

In this section, we will look at software that supports parallel computing. Since by parallel computing we mean symmetric multi-processing, i.e. the fact that many threads act in parallel at the same hierarchial level, this excludes client-server methods such as CORBA, Jini, or Java RMI although they can be tweaked to be used for symmetric multi-processing. Aspects of this will be discussed in Sec. 3.7.

Sometimes, parallelism can be achieved by just calling the same executable many times, each time with different parameters. This is for example common in Monte Carlo simulations in statistical physics, where the only difference between two runs is the random seed, or for sensitivity analysis for example in construction, where one is interested in how a system changes when any of a large number of parameters is changed from its base value. In multi-agent traffic simulations, one could imagine starting a separate activity planner, route planner, etc., for each individual agent. If each individual job runs long enough, the Operating System can be made responsible for job migration: If the load on one computer is much higher than the load on another computer, then a job may be migrated. Support for this does not only exist for shared memory machines (where it is taken for granted), but also for example for Linux clusters (Mosix, www page, accessed 2003).

Figure 5: Domain decomposition of a simulation of Switzerland. Different colors denote domains belonging to different CPUs.



A disadvantage of the Mosix-approach is that there is very little control. Essentially, one starts all jobs simultaneously, and the Operating System will distribute the jobs. However, if there are many more jobs than CPUs, this is not a good approach, since there will still remain large numbers of jobs per CPU, slowing down the whole process. In addition, there is no error recovery if one CPU fails. For that reason, there is emerging software support for better organization of such parallel simulations. An example of this is the Opera project, which allows to pre-register all the runs that one wants to do, and which then looks for empty machines where it starts the runs, keeps track of results, and re-starts runs for certain parameters if results do not arrive within a certain time-out (OPERA www page, accessed 2003).

For our simulations in the traffic area, however, it unfortunately turned out that the amount of computation per agent is too small to make either Mosix or Opera a viable option: Too much time is spent on overhead such as sending the information back and forth. An alternative is to bunch together the requests for several agents, but this makes the design and maintenance of the code considerably more complicated again. This is sometimes called "fine-grained" vs. "coarse-grained" parallelism. Despite of the relative simplicity of this approach, we are not aware of a project within transportation simulation which exploits this type of parallelism systematically.

Mosix or Opera perform parallelization by having separate executables. An alternative is to perform parallelization within the code: If there are several threads in a code that could be executed simultaneously, this could be done by threads. In pseudo-code, this could look like

```
for ( all agents i ) {
    start_thread_for_agent_i ;
}
for ( all agents i ) {
    wait_for_completion_of_thread_i ;
}
```

Note that `start_thread` just starts the thread and then returns (i.e. continues). In contrast, `wait_for_completion` continues when the corresponding thread is completed. That is, if the corresponding thread is completed before `wait_for_completion` is called, then `wait_for_completion` immediately returns. Otherwise it waits. The result is that all threads execute simultaneously on all available CPUs, and

the code continues immediately once all threads are finished.

Such code runs best on shared memory machines using POSIX threads for instance. Similar technology can in principle be used for distributed machines but is less standard (e.g. JavaParty www page, accessed 2005). In addition, on distributed machines thread migration takes considerable time, so that starting threads on such machines only makes sense if they run for relatively long time (at least several seconds).

All the above approaches only work either on shared memory machines or if the individual execution threads run independently for quite some time (several seconds). These assumptions are no longer fulfilled if one wants to run the simulation of the physical system (traffic micro-simulation) on a system with distributed memory computers. In such cases, one uses **domain decomposition** and **message passing**.

Domain decomposition, as already explained earlier, means that one partitions space and gives each local area to a different CPU (Fig. 5). For irregular domains, such as given by the graphs that underly transportation simulations, a graph decomposition library such as METIS (METIS www page, accessed 2005) is helpful.

Once the domains are distributed, the simulation of all domains is started simultaneously. Now obviously information needs to be exchanged at the boundaries; in the case of a traffic simulation, this means vehicles/travelers that move from one domain to another, but it also means information about available space into which a vehicle can move and on which the driving logic depends.

Software support for this is provided by message passing libraries such as MPI (MPI www page, accessed 2005) and PVM (PVM www page, accessed 2005). Originally, MPI was targeted towards high performance supercomputing on a cluster of identical CPUs while PVM was targeted towards inhomogeneous clusters with different CPUs or even Operating System on the computational nodes. In the meantime, both packages have increasingly adopted the features of the other package so that there is now a lot of overlap. Our own experience is that, if one is truly interested in high performance, MPI still has an edge: Its availability on high-performance hardware, such as Myrinet or parallel high-performance computers, is generally much better. Recall that there are some problems using MPI with Java (Sec.3.2.7).

3.6.4 Load balancing

A critical issue of parallel algorithms is load balancing: If one CPU takes considerably longer than all others to finish its tasks, then this is obviously not efficient. With fine-grained parallelism, this is not an issue: With, say, 10^6 agents and $1'000$ CPUs, one would just send the first $1'000$ agents to the $1'000$ CPUs, and then each time the computation for an agent is finished on a CPU, one would send another agent to that CPU, until all agents are treated. As said above, unfortunately this is inefficient in terms of overhead, and it is much better to send $1'000$ agents to each CPU right away. In this second case, however, it could happen that some set of agents needs much more time than all other sets of agents.

In such cases, one sometimes employs adaptive load balancing, that is, CPUs which lag behind offload some of their work to other CPUs. In the case of iterative learning iterations, those iterations can sometimes be used for load balancing (Nagel and Rickert, 2001). It is, however, our experience that in most cases it is sufficient to run a "typical" scenario, extract the computing time contributions for each entity in the simulation (e.g. each link or each intersection), and then use this as the basis for all load balancing in subsequent runs. Clearly, if one scenario is very different from another one (say, a morning rush hour compared to soccer game traffic), then the load balancing may be different.

3.6.5 Performance

Important numbers for parallel implementations are real time ratio, speed-up, and efficiency:

- **Real time ratio (RTR)** – describes how much faster than reality the simulation is. For example, an RTR of 100 means that 100 minutes of traffic are simulated in 1 minute of computing time. This number is important no matter if the simulation is parallel or not.

- **Speed-up** – describes how much faster the parallel simulation is when compared to a simulation on a single CPU. For the single CPU algorithm one either uses the parallel algorithm running on a single CPU, or an algorithm specifically tailored to a single CPU system.
- **Efficiency** – is obtained by dividing speed-up by the number p of CPUs that were used.

These numbers are all related, but they carry different meanings. RTR is most relevant if one is interested in real-time predictions – here, a simulation needs to run many times faster than real time to be useful. Speed-up is relevant to compute how much faster the parallel application is. For example, a speed-up of 100 tells us that 4 days of computing can be reduced to one hour. Finally, efficiency describes how efficient the parallel CPUs are utilized – for example, an efficiency of 50% means that doing the same computation on a single CPU would be half as expensive in terms of cost per result if one was willing to wait accordingly.

Speed-up is limited by a couple of factors. First, the software overhead appears in the parallel implementation since the parallel functionality requires additional lines of code. Second, speed-up is generally limited by the speed of the slowest node or processor. Thus, we need to make sure that each node performs the same amount of work. i.e. the system is load balanced (see above). Third, if the communication and computation cannot be overlapped, then the communication will reduce the speed of the overall application. A final limitation of the speed-up is known as Amdahl's Law. This states that the speed-up of a parallel algorithm is limited by the number of operations which must be performed sequentially. Thus, let us define, for a sequential program, t_S as the amount of the time spent by one processor on sequential parts of the program and t_P as the amount of the time spent by one processor on parts of the program that can be parallelized. Then, we can formulate the serial run-time as $T(1) := t_S + t_P$ and the parallel run-time as $T(p) := t_S + t_P/p$. Therefore, the serial fraction F will be $F := t_S/T(1)$, and the speed-up $S(p)$ is

$$S(p) := \frac{T(1)}{T(p)} = \frac{t_S + t_P}{t_S + \frac{t_P}{p}} = \frac{1}{F + \frac{1-F}{p}} \; .$$

This means that even for $p \to \infty$, the speed-up can be no larger than $1/F$.

As an illustration, let us say we have a program of which 80% can be done in parallel and 20% must be done sequentially. Then even for $p \to \infty$, we have $S(p) = 1/F = 5$, meaning that even with an infinite number of processors we are no more than 5 times faster than with a single processor.

For multi-agent transportation simulations, the strategy computing modules are easy to parallelize since all agents are independent from each other. In the future, one may prefer to generate strategies at the household level, but that still leaves enough parallelism. More demanding with regards to parallel performance would be inter-household negotiations such as for social activites or for ride sharing. This is, however, virtually unexplored even in terms of modeling, let alone implementation.

This leaves the MobSim. In the past, this was considered the largest obstacle to fast large-scale multi-agent transportation simulations. It is, however, now possible to run systems with several millions of agents about 800 times faster than real time, meaning that 24 hours of (car) traffic in such a system can be simulated in about 2 minutes of computing time (Fig. 6). The technology behind this is a 64 CPU Pentium cluster with Myrinet communication (Cetin and Nagel, 2003); the approximate cost of the whole machine (excluding file server) is about 200 kEuro – expensive, but still affordable when compared to the 100 MEuro that a typical supercomputer costs. Tab. 1 compares the computing speeds of several exisiting approaches. One sees that a simulation using Ethernet for communication levels out at an RTR of 166. The reason for this is the bad latency of standard Ethernet: With a latency of about 0.5 msec, an average of six neighbors, and two messages exchanges per time step, latency alone uses 6 msec per time step. In consequence, the simulation can at most process $1 \text{ sec}/6 \text{ msec} \approx 166$ time steps per second. The superlinear speed-up is due to the fact that the scenario did not fit into the 1 GByte memory that was available per computational node.

These results make clear that in terms of computing speed the traffic simulation, or what is called the "network loading" in dynamic assignment, is no longer an issue (although considerable work will still be necessary to fine-tune the achievements and make them robustly useful for non-experts in the area). This means that, at least in principle, it is now possible to couple agent-based travel behavior models on the

Figure 6: Computational performance of a parallel queue MobSim for all of Switzerland. LEFT: Real Time Ratio (RTR), i.e. how many times faster than reality the simulation runs. RIGHT: Speed-up, i.e. how many times faster than the single-CPU version the simulation runs.

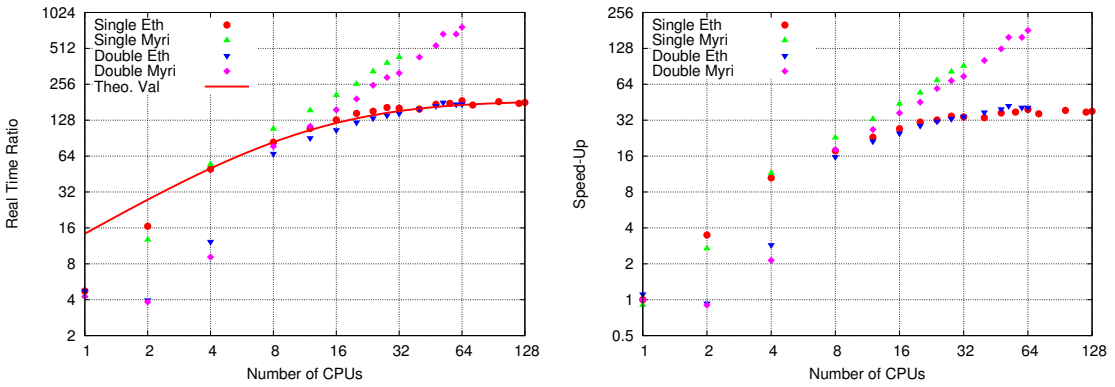


Table 1: Computational performance results for parallel MobSim.

Project	Best RTR	Best Speedup	CPUs	Vehs	Links	Remarks
PARAMICS[a]	1.8	62	64	384 k	320[b]	SGI R10000 MIPS 250Mhz
TRANSIMS (standard)[c]	10	2.5	32	100 k	20k	PCs
TRANSIMS (tuned)[d]	65	16	32	100 k	20k	PCs
AIMSUN[e]	4	3.5	8	10k	1.5k	Sun and PCs, threads
DYNEMO[f]	6	15	19	250 k	14 k	PCs
METROPOLIS[g]	34	1.3	2	1'000 k	30k	PCs, (hyper)threads
QSIM[h]	770	180	64	1'000 k	30k	PCs, Myrinet

[a] Quadstone Limited (2002)
[b] Number approximate; small benchmarking network
[c] Our own tests
[d] (Nagel and Rickert, 2001)
[e] (Barceló *et al.*, 1998a,b)
[f] Nökel and Schmidt (2000)
[g] Our own tests using a Pentium 4, 3Ghz, hyperthread single-CPU (2 virtual CPUs)
[h] Cetin and Nagel (2003)

strategy level with an agent-based MobSim on the physical level. This should make it possible to have consistent completely agent-based large scale transportation simulation packages within the foreseeable future.

3.6.6 Summary and discussion of "Parallel computing"

Using MPI and C++ for parallel computing is a mature and stable technology for parallel computing and can be recommended without reservations. Diligent application of these technologies makes very fast parallel MobSims – 800 times faster than real time with several million travelers – possible. This has so far been demonstrated with simple traffic dynamics (the queue model; Cetin and Nagel, 2003); the fastest parallel simulation with a more complicated traffic dynamics is, as far as we know, TRANSIMS with a maximum published computational speed of 65 times faster than real time (Nagel and Rickert, 2001). This implies that a realistic and fast enough agent-based traffic simulation, if this is desired, is feasible.

In addition, running the strategy computations for several million agents is expected to be easy to paral-

lelize since, given current methods, those computations are essentially independent except at the household level. As was discussed, future methods will probably incorporate inter-household interaction already at the strategy level (leisure; ride sharing), which will make this considerably more challenging.

A remaining problem is the coupling of parallelized modules. As long as the modules are run sequentially, several good methods are available to solve the problem. Sequential modules imply that as long as one module is running, no other modules can interfere with changes. For example, travelers cannot replan routes or schedules as long as the MobSim is running. Once one wants to loosen that restriction and allow within-period replanning, then designing an efficient and modular parallel computing approach remains a challenge. This will be discussed in the next section.

3.7 Distributed computing

3.7.1 Introduction

The last section, on parallel computing, has explored possibilities to accelerate single modules by parallelizing them. Module coupling in a parallel environment was not discussed. This will be done in this section. Note that Sec. 3.5 already already discussed aspects of how to couple modules that run potentially on different computers. However, in that section it was still assumed that the modules would run sequentially, not simultaneously. Running different modules simultaneously is the new aspect that will be explored in this section.

In fact, the reason for this section on distributed computing is that one would want to combine within-period replanning with parallel computing: within-period replanning is needed for modeling reasons, and parallel computing is needed in order to run large scenarios, both for computational speed and for memory reasons. It was already said, in Sec. 3.5, that subroutine calls or its remote variants achieve within-period replanning, but because everything is run sequentially, this is slow. Parallel computing, in Sec. 3.6, discussed how to speed up the sequential computation, but did not discuss within-period replanning. This section will discuss how these things could be combined. Two technologies will be discussed:

- Calling strategic subroutines from within a parallel MobSim.
- Distributed agents.

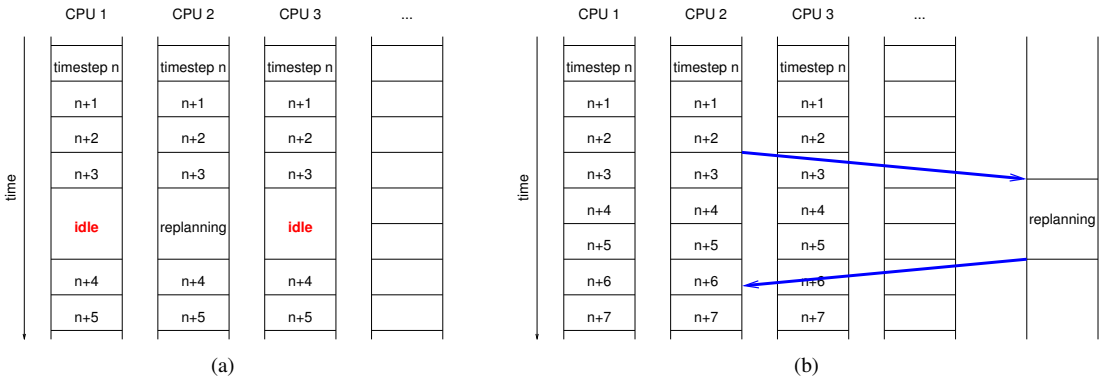
3.7.2 Subroutine calls from a parallel MobSim

As for sequential codes, the first technology that comes to mind to enable within-period replanning is to use, from inside a parallel MobSim, the "subroutine call" technique from Sec. 3.5.2 or its "remote" variant from Sec. 3.5.4. This would probably be a pragmatically good path to go; however, the following counter-arguments need to be kept in mind:

- Using subroutine calls for the modules does again not solve the incompatibility problems discussed in Sec. 3.5.2.
- Using CORBA is feasible but relatively complicated.
- Using Java RMI solves the incompatibility problem only in part: Different operating systems can interact, but all programs need to be written in Java. In particular, the parallel MobSim would have to be written in Java, and not standardized MPI support is available for Java as discussed in Sec. 3.2.7.
- Calling the strategy subroutines from the MobSim causes load-balancing problems as shown in Fig. 8(a): If only one CPU works on strategy computation instead of on the MobSim, all other CPUs have to wait.

- Finally, using subroutine calls implies that all the agent information is in the MobSim so that it can be passed as an argument. However, a complete agent can be a heavyweight and complicated structure, containing information on many levels from routes to demographics, or more than one strategy (Sec. 2.4). Having a parallel MobSim implies that all that information needs to be "carried around" via the message passing system. Intuitively, this does not seem the correct approach. And in fact, the database approaches in Sec. 3.5.6 imply that it should be possible to keep most of the agent information outside the parallel MobSim. This is discussed below.

Figure 7: Parallel implementation of within-period replanning. (a) Load balancing problem if strategy computation is a subroutine call from within the parallel MobSim. (b) Load balancing problem replaced by "delayed response" if strategy computation is done via external server rather than via internal subroutine call.



3.7.3 Distributed agents concept

Conceptually, it should be possible to design a simulation package similar to Fig. 1: There could be a parallel MobSim in which the physical reality ("embodiment") of the agents is expressed, and then there could be many independent "agent brains" in which the agents, independently from each other, receive stimuli from the MobSim, compute strategies, and send them back to the MobSim.

For example, once an agent arrives at an activity location, that information is sent to the agent brain-module. That brain-module then decides how long the agent will stay at that activity. It will then, at the right point in time, send information to the MobSim that the agent wants to leave the activity location and what it means of transportation is going to be.

Similarly, an agent being stuck in a traffic jam would send that information to the brain-module. The brain-module would compute the consequences of this, and decide if changes to the original plan are necessary. For example, it could decide to take an alternate route, or to adjust the activity schedule. By doing this, many CPUs could be simultaneously involved in agent strategy generation; in the limit, there could be as many strategy generating CPUs as there are agents.

This idea is similar to robot soccer, where teams of robots play soccer against each other. In robot soccer, robots typically have some lower level intelligence on board, such as collision avoidance, but larger scale strategies are computed externally and communicated to the robot via some wireless protocol. The analogy with our simulations is that in our case the "embodiment" (where the robots collide etc.) is replaced by the MobSim. The difference with robot soccer is that there is no cooperative decision making (at least in the simplest version of MATSIM).

Such a distributed approach replaces the load-balancing problem by a "delay problem" (Fig. 8(b)): The distributed implementation introduces a delay between the request for a new strategy and the answer by the strategy module; in the mean time, the physical world (represented by the MobSim) advances. In consequence, the distributed approach can only be useful if the simulation system can afford such a delay. Fortunately, such a delay is plausible since both humans and route guidance services need

some time between recognizing a problem and finding a solution. However, care needs to be taken to implement the model in a meaningful way; for example, there could be a standard delay between request and expected answer, but if the answer takes longer, then the MobSim is stopped anyway.

Despite these caveats, we find the idea of autonomous computational agents very attractive, but with current software and hardware tools, it seems out of reach. Even if we could run 1000 independent processes on a single CPU (which seems doubtful), a simulation with 10 millions of agents would still need 10 000 CPUs for strategy computations. For the time being, we can only plan to have a few brain-modules per CPU that are each responsible for a large number of agents each.

3.7.4 Distributed agent software

Given the distributed agent idea, the question becomes which software to use. At this point, there is a lot of talk about this kind of approach, under the name of, say, "software agents" or "peer-to-peer computing", but there are no established standards and most of today's development in that field are geared by interoperability concerns without much regard about performance.

The main problem, when compared to the remote procedure calls discussed in Sec. 3.5.4, is that for a distributed computing approach one wants to have the modules run simultaneously, not sequentially. A subroutine call, even when remote, has, however, the property that it waits for the call to return; or alternatively, it does not wait but then there is also no default channel to communicate the results back. One way out is to use threads (Sec. 3.6.3) combined with remote procedure calls, in the following way:

```
...
for ( all agents i ){
    start_thread_for_agent(i) ;
}
for ( all agents i ) {
    wait_for_completion_of_thread(i) ;
}
...
```

This is the same is in Sec. 3.6.3. In contrast to there, however, the thread itself would now just be a remote procedure call:

```
thread_for_agent(i) {
    invokeRemoteTask(...) ;
}
```

In this way, the local parallelism that threads offer could be combined with the interoperability that remote procedure calls offer.

Although this combined threads/RPC approach should work in any modern computer language, at this point, it looks like Java may become one of the emerging basic platforms on which software agents are built. Potential technology such as Jini may help to build this architecture. Jini is essentially an automatic registration mechanism for software agents, built as an application layer on top of different communication mechanisms such as Java RMI, CORBA or XML. The purpose of Microsoft .NET is similar (i.e. self-registering of web services) but it is multi-language, single OS (Windows based), and it favors XML for the communication layer.

3.7.5 Coupling the parallel MobSim with distributed agents

Even if Java becomes the emerging standard for software agents, we have a problem of how to couple this to the MobSim. Here are several options, with their advantages and disadvantages:

Table 2: Comparison of computational techniques.

Module coupling via ...	General problems	Additional problems when the MobSimis parallel
... subroutine calls	Coupling between different prog languages / OS difficult	Load balancing, Fig. 8(a)
... CORBA	Complexity of the code	Load balancing, Fig. 8(a)
... Java RMI	Restricted to Java	Load balancing, Fig. 8(a)
... files	Essentially restricted to period-to-period replanning	
... database	Within-period replanning difficult (but probably possible)	DB will be performance bottleneck; distributed DB difficult. Mechanism for within-period replanning unclear
... messages	Difficult, technology not established	"Delayed response", Fig. 8(b)

- One option would be, once more, to write the MobSim itself in Java. A problem, as discussed in Sec. 3.2.7, is that there is not standardized MPI support for Java.
- Another option would be to write everything in C++ together with MPI. This option could not benefit from any Java-based technology, but it might be the fastest path to a working system. It would have the disadvantage that it would essentially be restricted to Unix (although there are MPI implementations for Windows). In addition, it is restricted by the limitations imposed by MPI. One of those restrictions is that it is difficult if not impossible to have "self-registering modules". By this we mean a system where all modules are started asynchronously and where they establish their respective existence plus the communication channels by themselves rather than by design. For example, a new router could be added to the system, it would broadcast its capabilities, and the simulation could incorporate it while it is running. Such automatic registration techniques would simplify the running of distributed simulations as well as the prototyping of new modules.
- Another option is to leave the MobSim in C++/MPI, but use Java on the strategy level. The open question is how to couple those two levels; one option is to use MPI in spite of the non-standard MPI implementation for Java. This is in fact possible, but differences of the internal representation of data items between C++ and Java need to be considered (Cetin, 2005).
- Given all the different drawbacks with the existing and emerging technologies, another option is to write the communication library yourself (Gloor, 2001, 2005). For example, one can use TCP (Transmission Control Protocol) or UDP (User Datagram Protocol) directly. TCP enables reliable communication, resending packets until safe arrival is confirmed; UDP sends packets out and then forgets about them. The disadvantage of TCP is that each computational node needs to maintain connections with each other computational node that it wants to talk to, leading to a large number of communication channels that need to be maintained. All this is not necessary with a UDP-based communication, which, however, has the disadvantage that messages can get lost and which in consequence makes any distributed implementation extremely difficult to debug, and also difficult to use since one needs to distinguish between effects caused by the simulated dynamics, and effects caused by the computational implementation. Nevertheless, the UDP-implementation is closer to peer-to-peer communication systems , which are a topic of much current research, and which also do *not* assume reliability. Therefore, a simulation technology designed with UDP might be able to take advantage of peer-to-peer research results, which may not be possible for a simulation technology designed with TCP.

3.7.6 Summary of "Distributed computing"

Tab. 2 summarizes the advantages and disadvantages of the parallel and distributed approaches discussed in this test. Overall, one can maybe say that the approach of distributed software agents, as visualized by Fig. 1, is very promising but also very experimental. Besides the experimental stage in the area of software agents alone, it is unclear which technology is best able to couple the software agents on the strategy level with the physical MobSim which uses computational physics techniques.

In the meantime, there will be single-CPU implementations of agent technology. They will help understanding the issues of agent-based computation of travel behavior, but it is unclear if they will be able to simulate large scale scenarios within acceptable computing time.

3.8 Other issues

3.8.1 GRIDS

GRIDS are currently pushed in particular by the high energy physics community, which will very soon produce much more experimental data than the computers at the experimental site can process. Therefore, a world-wide system is implemented, which will distribute both the computational load and the data repositories.

When looking at GRIDS, one has to notice that they are as much a political decision as a technological one. Clearly, it would have been much easier to move all the computers to the experimental site and to construct one system there which would serve the whole world, rather than distributing the whole system. Given current funding systems, it is, however, impossible to do this, since the economy around the experimental location would be the only economic benefactor.

In consequence, when looking at computational GRIDS to solve computational transportation problems, one first needs to consider if a problem is truly large enough to not run on a single site. Since the answer to this question is probably nearly always "no", the primary reason which leads to computational GRIDS in other areas of science does not apply to transportation.

There are, however, other reasons why such an architecture may pay off in transportation. The foremost reason probably is that all data in the geosciences is derived and maintained locally: Counties have their very detailed but very local databases, regions have data for a larger region but less detailed, etc. In addition, there are vast differences between the quality of data and the implementation in terms of Operating System, database vendor, and database structure.

For a large variety of reasons, it will probably be impossible to ever move all this data into a centralized place. Therefore, it will be necessary to contruct a unified interface around all these different data sources. For example, it should be possible to request U.S. and Swiss census data with the same commands as long as the meaning of the data is the same, and if the meaning is different, there should be clear and transparent conversion standards. The advantage of such a standardized interface would be that codes would become easily transferable from one location to another, facilitating comparison and thus technological progress. This would use the database aspect of the GRID technology rather than the computing aspect of the GRID technology.

3.8.2 Numerical analysis

Another aspect of "computation for travel behavior research" concerns the area of numerical analysis, e.g. non-linear optimization, matrix inversion, etc. Besides as solutions to route assignment, such approaches are for example used for OD matrix corrections from traffic counts, for model calibrations, for the estimation of random utility models, etc. These approaches, as important as they are, are beyond the scope of this paper. They would necessitate a separate review paper.

3.8.3 Visualization/Virtual reality

Before coming to an end, let us briefly touch upon the subject of visualization. It is by now probably clear to many researchers that communication of scientific and technical results to people outside the field of transportation research is easiest done by the means of visualization. It is therefore encouraging to know that we are close to solutions which will bring certain aspects of virtual reality to a laptop. This will make it possible to bring both scenario results and the means to look at them to any place of the world where

they are needed. We see this as a very positive outcome since it is, in our opinion, the citizen who needs to decide about the transportation infrastructure.

4. Summary

Within the context of travel behavior research, many approaches look at individual travelers as the principal unit of research. This holds true for example for random utility models, which predict behavioral probabilities as a function of individual demographic and individual option-dependent characteristics, but also for psychological/rule-based approaches. Using these approaches for transportation policy will only be possible if those results can be put into a framework where it applies to real-world transportation-related questions. One viable method to achieve this goal is multi-agent simulations, which can treat several millions of individual agents within a consistent simulation framework. The agent-based approach has the advantage that it can directly incorporate behavioral research results; the same is not true for the 4-step process or any similar aggregated approach. It is thus possible to program a multi-agent simulation with the rules and results of travel behavior research and thus obtain a simulation system for policy forecasting that is based on the research results.

The first part of this paper explains how such a simulation system could be designed. There is increasing consensus in the artificial intelligence community that in order to understand intelligence it is necessary to simulate explicitly the interaction between the physical world and the space of strategy formation. This "embodiment hypothesis" is, in the context of travel behavior research, best put into practice by having the strategy generation separated from the MobSim, which receives the strategies of the agents, executes them, and feeds back information about the agent's actual experiences (such as incurred travel times). Besides strategy modules and a MobSim, a functional system needs initial/boundary conditions, and a learning method.

The second, considerably longer, part of this paper then discusses computational and implementation methods. Among the discussed areas are the application of computer science search methods including evolutionary algorithms to travel behavior modeling, or capabilities that databases could and should have to be even more useful. Special emphasis is given to the trade-offs between ease-of-programming, interoperability, and computational performance. In particular, we expect that in the future much progress will be made by coupling modules coming from different research groups, which will mean that codes using different programming languages running on different operating systems need to be coupled. The paper attempts to give some guidance plus pointers to relevant methods for these cases, and how they relate to performance-oriented parallel computing. The overall conclusion is that many of the necessary aspects of inter-operable software construction are just emerging. Thus, there is hope that the necessary methods will eventually become available, but at this point, many aspects are still rather experimental. In contrast, as long as modules run sequentially (that is, within-period replanning is excluded), very helpful standardized methods such as XML, CORBA, Java RMI, or MPI are now available.

The paper deliberately concentrates on fairly technical questions, since such aspects are potentially important for large scale implementations, and they are rarely discussed in sufficient detail. In particular, the computational methods part of the paper is meant as a compendium for someone considering to implement parts of such a simulation package. Overall, it is hoped that this paper stimulates discussion about how travel behavior research models can be implemented, and which features may be desirable to make them inter-operable.

Acknowledgments

We have benefited from many inspiring presentations and discussions within the Department of Computer Science and within the CoLab. Without that stimulating environment, a similar overview over computational methods would not have been possible.

References

Arentze, T., F. Hofman, H. van Mourik and H. Timmermans (2000) ALBATROSS: A multi-agent rule-based model of activity pattern decisions, *Paper*, **00-0022**, Transportation Research Board Annual Meeting, Washington, D.C.

Astarita, V., K. Er-Rafia, M. Florian, M. Mahut and S. Velan (2001) A comparison of three methods for dynamic network loading, *Transportation Research Record*, **1771** 179–190.

Axhausen, K. (1990) A simultaneous simulation of activity chains, in P. Jones (Ed.), *New Approaches in Dynamic and Activity-based Approaches to Travel Analysis*, 206–225, Avebury, Aldershot.

Balmer, M., M. Bernard and K. Axhausen (2005) Matching geo-coded graphs, in *Proceedings of Swiss Transport Research Conference (STRC)*, Monte Verita, CH, URL `www.strc.ch`.

Barceló, J., J. Ferrer, D. Garcia, M. Florian and E. Le Saux (1998a) Parallelization of microscopic traffic simulation for ATT systems, in P. Marcotte and S. Nguyen (Eds.), *Equilibrium and advanced transportation modelling*, 1–26, Kluwer Academic Publishers.

Barceló, J., J. Ferrer, D. García and R. Grau (1998b) Microscopic traffic simulation for att systems analysis. a parallel computing version, Contribution to the 25th Aniversary of CRT, University of Montreal. See www.tss-bcn.com/documents.html.

Barrett, C. L. and C. M. Reidys (1997) Elements of a theory of simulation I: sequential CA over random graphs, *Los Alamos Unclassified Report (LA-UR)*, **97-2343**, Los Alamos National Laboratory.

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

Ben-Akiva, M. and S. R. Lerman (1985) *Discrete choice analysis*, The MIT Press, Cambridge, MA.

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

Boyce, D., D. Lee, B. Janson and S. Berka (1997) Dynamic route choice model of large-scale traffic network, *Journal of transportation engineering-asce*, **123** (4) 276–282.

Cascetta, E., A. Nuzzolo, F. Russo and A. Vitetta (1996) A modified logit route choice model overcoming path overlapping problems: Specification and some calibration results for interurban networks, in *Proceedings of the 13th International Symposium on Transportation and Traffic Theory*, 697–712, Lyon, France.

Cetin, N. (2005) *Large-scale parallel graph-based simulations*, chap. 5–7, Ph.D. thesis, ETH Zürich, Switzerland. Also see www.vsp.tu-berlin.de/publications.

Cetin, N. and K. Nagel (2003) A large-scale agent-based traffic microsimulation based on queue model, in *Proceedings of Swiss Transport Research Conference (STRC)*, Monte Verita, CH, URL `www.strc.ch`. Earlier version, with inferior performance values: Transportation Research Board Annual Meeting 2003 paper number 03-4272.

Chabini, I. (1998) Discrete dynamic shortest path problems in transportation applications: Complexity and algorithms with optimal run time, *Transportation Research Record*, **1645** 170–175.

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

CORBA www page (accessed 2005) CORBA: Common Object Request Broker Architecture, URL `www.corba.org`.

CVS www page (accessed 2005) CVS: Concurrent versions (control) system, URL `www.cvshome.org`.

Cygwin www page (accessed 2005) Cygwin: A Linux-like environment for Windows, URL `www.cygwin.com`.

Daganzo, C. (1998) Queue spillovers in transportation networks with a route choice, *Transportation Science*, **32** (1) 3–11.

de Palma, A., C. Fontan, F. Marchal, O. Mekkaoui, K. Motamedi and O. Sanchez (2001) Final deliverable of the QUATUOR project, French ministry of transport, *Tech. Rep.*, **DRAST/PREDIT 98MT30**, University of Cergy-Pontoise.

de Palma, A. and F. Marchal (2001) Dynamic traffic analysis with static data: some guidelines from an application to Paris, *Transportation Research Record*, **1756** 76–83.

de Palma, A. and F. Marchal (2002) Real case 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 developement of a unified modelling framework for the household activity-travel scheduling process, in *Verkehr und Mobilität*, vol. 66 of *"Stadt Region Land"*, Institut für Stadtbauwesen, Technical University, Aachen, Germany.

Dorigo, M., G. DiCaro and L. Gambardella (1999) Ant algorithms for discrete optimization, *Artificial Life*, **5** (2) 137–172.

Doxygen www page (accessed 2005) Doxygen: A documentation system, URL `www.doxygen.org`.

DYNAMIT www page (accessed 2005) URL `mit.edu/its`.

DYNASMART www page (accessed 2005) URL `www.dynasmart.com`.

Ferber, J. (1999) *Multi-agent systems. An Introduction to distributed artificial intelligence*, Addison-Wesley.

Friedrich, M., I. Hofsäß, K. Nökel and P. Vortisch (2000) A dynamic traffic assignment method for planning and telematic applications, in *Proceedings of Seminar K*, European Transport Conference, Cambridge, GB.

Gawron, C. (1998a) An iterative algorithm to determine the dynamic user equilibrium in a traffic simulation model, *International Journal of Modern Physics C*, **9** (3) 393–407.

Gawron, C. (1998b) Simulation-based traffic assignment, Ph.D. thesis, University of Cologne, Cologne, Germany. Available via www.zaik.uni-koeln.de/~paper.

Gloor, C. (2001) Modelling of autonomous agents in a realistic road network (in German), Diplomarbeit, Swiss Federal Institute of Technology ETH, Zürich, Switzerland.

Gloor, C. (2005) Distributed intelligence in real-world mobility simulations, Ph.D. thesis, Swiss Federal Institute of Technology ETH.

Goldberg, D. (1989) *Genetic Algorithms in Search, Optimization and Machine Learning*, Addison-Wesley.

Hofbauer, J. and K. Sigmund (1998) *Evolutionary games and replicator dynamics*, Cambridge University Press.

INRO Consultants Inc. (1998) Emme/2 user's manual, release 9.0.

Java 3D API www page (accessed 2005) URL `java.sun.com/products/java-media/3D`.

Java Generics www page (accessed 2005) URL `developer.java.sun.com/developer/technicalArticles/re`

Javadoc www page (accessed 2005) Javadoc: A tool for api documentation, URL `java.sun.com/j2se/javadoc`.

JavaParty www page (accessed 2005) Javaparty: A distributed companion to java, URL `www.ipd.uka.de/JavaParty`.

Kaufman, D. E., K. E. Wunderlich and R. L. Smith (1991) An iterative routing/assignment method for anticipatory real-time route guidance, *Tech. Rep.*, **IVHS Technical Report 91-02**, University of Michigan Department of Industrial and Operations Engineering, Ann Arbor MI 48109, May 1991.

Kitamura, R. (1996) Applications of models of activity behavior for activity based demand forecasting, in *TMIP (Travel model improvement program) Activity-based travel forecasting conference*, June 1996, URL `tmip.fhwa.dot.gov/clearinghouse/docs/abtf/`.

Kurman, C. and T. Stricker (1999) A comparison of three gigabit technologies: Sci, myrinet and sgi/cray t3d, in H. Hellwagner and A. Reinefeld (Eds.), *SCI: Scalable Coherent Interface, Architecture and Software for High-Performance Compute Clusters*, vol. 1734 of *Lecture Notes in Computer Science*, 39–68, Springer, ISBN 3-540-66696-6.

Lohse, D. (1997) *Verkehrsplanung*, vol. 2 of *Grundlagen der Straßenverkehrstechnik und der Verkehrsplanung*, Verlag für Bauwesen, Berlin.

Marchal, F. (2003) Implementation of spill-back effects in event-based traffic simulations, D-INFK Seminar, ETH Zürich, June 2003.

METIS www page (accessed 2005) URL `www-users.cs.umn.edu/~karypis/metis`.

MPI www page (accessed 2005) URL `www-unix.mcs.anl.gov/mpi/`. MPI: Message Passing Interface.

MYSQL www page (accessed 2005) MYSQL: An open-source SQL database, URL `www.mysql.com`.

Nagel, K. and M. Rickert (2001) Parallel implementation of the TRANSIMS micro-simulation, *Parallel Computing*, **27** (12) 1611–1639.

Nökel, K. and M. Schmidt (2000) Parallel DYNEMO: Mesoscopic traffic flow simulation on large networks, preprint.

OpenGIS www page (accessed 2005) URL `www.opengis.org`.

OPERA www page (accessed 2003) OPERA: process management for distributed, heterogeneous environments, URL `www.inf.ethz.ch/department/IS/iks/research/opera.html`.

ORACLE www page (accessed 2005) Oracle database server, URL `www.oracle.com/products`.

Ortúzar, J. d. D. and L. Willumsen (1995) *Modelling transport*, Wiley, Chichester.

Perko, A. (1986) Implementation of algorithms for k shortest loopless paths, *Networks*, **16** 149–160.

PVM www page (accessed 2005) PVM: Parallel Virtual Machine, URL `www.epm.ornl.gov/pvm`.

Quadstone Limited (2002) Quadstone paramics v1.4, performance benchmarks, URL `www.paramics-online.com`.

Raney, B. and K. Nagel (2002) Iterative route planning for modular transportation simulation, in *Proceedings of Swiss Transport Research Conference (STRC)*, Monte Verita, CH, URL `www.strc.ch`.

Raney, B. and K. Nagel (2003) Truly agent-based strategy selection for transportation simulations, *Paper*, **03-4258**, Transportation Research Board Annual Meeting, Washington, D.C.

Repast www page (accessed 2005) Recursive porous agent simulation toolkit, URL `repast.sourceforge.net`.

RMI www page (accessed 2005) Java RMI: Java Remote Method Invocation, URL `java.sun.com/products/jdk/rmi`.

Schwerdtfeger, T. (1987) Makroskopisches Simulationsmodell für Schnellstraßennetze mit Berücksichtigung von Einzelfahrzeugen (DYNEMO), Ph.D. thesis, University of Karsruhe, Germany.

Sheffi, Y. (1985) *Urban transportation networks: Equilibrium analysis with mathematical programming methods*, Prentice-Hall, Englewood Cliffs, NJ, USA.

Stein, D. and others (Eds.) (several volumes, since 1988) *Lectures in the sciences of complexity*, Santa Fe Institute in the sciences of complexity, Addison-Wesley.

Swarm www page (accessed 2005) SWARM, a software package for multi-agent simulation of complex systems, URL `www.swarm.org`.

TSP challenge www page (accessed 2003) 8th DIMACS implementation challenge: The traveling salesman problem, URL `www.research.att.com/~dsj/chtsp/`.

TSPLIB www page (accessed 2003) TSPLIB, a library of sample instances for the tsp (and related problems), URL `www.iwr.uni-heidelberg.de/groups/comopt/software/TSPLIB95/`.

URBANSIM www page (accessed 2003) URL `www.urbansim.org`.

Waddell, P., A. Borning, M. Noth, N. Freier, M. Becke and G. Ulfarsson (2003) Microsimulation of urban development and location choices: Design and implementation of UrbanSim, *Networks and Spatial Economics*, **3** (1) 43–67.

Waddell, P., H. Ševcíková, D. Socha, E. Miller and K. Nagel (2005) OPUS: An international collaboration to develop an open platform for urban simulation, *Paper*, **428**, 9th Conference on Computers in Urban planning and urban management (CUPUM), University College London, UK. See www.cupum.org.

www page, M. (accessed 2003) MOSIX, a cluster management system, URL `www.mosix.org`.

XML www page (accessed 2005) XML: eXtensible Markup Language, URL `www.w3.org/XML`.

Zuylen, H. v. and H. Taale (2004) Urban networks with ring roads: A two-level, three player game, *Paper*, **04-1659**, Transportation Research Board Annual Meeting, Washington, D.C.