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

David Charypar, IVT, ETH Zurich
Kay W. Axhausen, IVT, ETH Zurich
Kai Nagel, VSP, TU Berlin

Conference paper
Session 5 Simulating behaviors of individuals, households, and organizations

The Expanding Sphere of Travel Behaviour Research

11th International Conference on Travel Behaviour Research
Kyoto, 16-20 August 2006

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

David Charypar
IVT
ETH Zurich
Switzerland
Phone: +41 44 633 35 62
Fax: +41 44 633 10 57
E-mail: charypar@ivt.baug.ethz.ch

Kay W. Axhausen
IVT
ETH Zurich
Switzerland

Kai Nagel
VSP
TU Berlin
Germany

Abstract

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

Keywords

Daily plan optimization, agent-based travel behavior simulation, evolution strategy

Preferred citation style

Charypar, David, Axhausen, Kay W., and Nagel, Kai (2006) Implementing Activity-Based Models: Accelerating the Replanning Process of Agents Using an Evolution Strategy, paper presented at the 11th International Conference on Travel Behaviour Research, Kyoto, August 2006.

1. Introduction

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

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

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

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

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

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

The rest of this paper is structured as follows: In section 2 we describe the overall design of

our agent-based microsimulation of daily demand, in section 3 we introduce and explain the measures taken to speed up this simulation system, in section 4 we present the test scenario that we use to compare the original and the improved system, in section 5 we discuss our results, and we conclude and give an outlook to future work in section 6.

2. The Microsimulation Toolkit

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

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

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

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

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

Balmer *et al.* (2006)

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

3. Speeding up MATSIM-T

In this section we present the approaches taken to speed up MATSIM-T.

An important part of our work is to reduce the time needed for our simulation toolkit to converge to a point of rest at which all the agents are executing a daily plan that they cannot improve significantly themselves. To reduce this time there are a couple of approaches that one can take. On way is to reduce the number of iterations of the whole system that are needed to find a relaxed state. This can be done for instance by improving the replanning module to produce better plans for the share of agents that are replanning. Another way one can go is to find the optimal percentage of agents selected for replanning in a particular iteration. A completely different approach to improve the overall performance of the system is to reduce the time needed to process a single iteration. This can be done by optimizing the individual modules. On overview of the improvements that were done during the work described in this paper can be seen in Figure 1.

3.1 Producing Better Plans

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

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

So far, the optimization algorithm used in the replanning module of MATSIM-T was a specially made genetic algorithm (GA)(see Meister *et al.* (2005)). Although it's performance was generally sufficient to solve the optimization problem at hand we assume that a more sophisticated algorithm would yield better replanned daily plans or at least provide them with less computa-

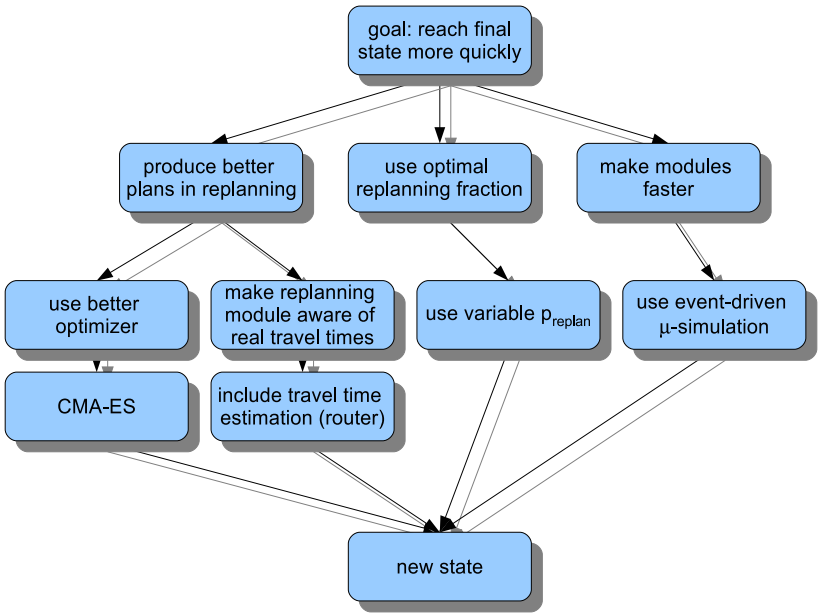


Figure 1: Steps that were implemented to speed up the MATSIM toolkit

tional effort. We chose to use the covariance matrix adaptation evolution strategy (CMA-ES) as described in Hansen and Kern (2004) to replace our current GA.

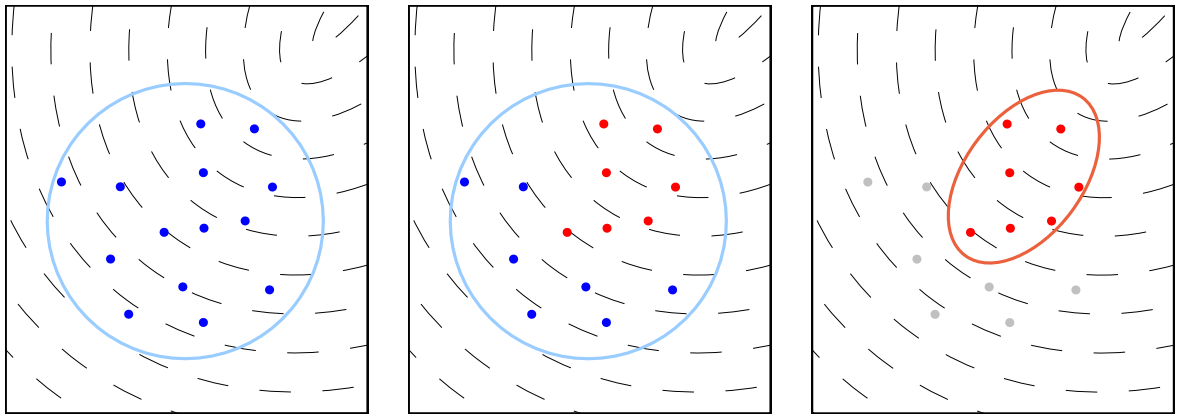
Evolution strategies belong to the field of evolutionary computation. In general, they represent stochastic population based optimization algorithms for continuous space problems that use recombination and mutation operators to produce new candidate solutions.

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

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

3.1.2 Description of CMA-ES

The covariance matrix adaptation evolution strategy (CMA-ES) is an iterative stochastic population based optimization algorithm. It holds a population of candidate solutions where each candidate solution is a point in n-dimensional vector space. n is referred to as problem dimension lying in the range of 3 to 6 in our case. CMA-ES also stores a sampling distribution



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

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

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

Figure 2: The adaptation of the sampling distribution in CMA-ES

(consisting of a center point and a covariance matrix) that is modified during the optimization process. In every generation a new population of sampling points is generated, the best candidates are selected and they are used to modify the sampling distribution accordingly. Running CMA-ES consists of the following steps:

1. Initialize the center of the sampling distribution (this corresponds to the starting point) and the global step size. Initialize the covariance matrix to an identity matrix.
2. Sample the objective function using N sampling points according to the current sampling distribution (consisting of center, covariance matrix and global step size). N is a parameter of the algorithm and was set to default values according to Hansen and Kern (2004).
3. Evaluate the objective function on all sampling points.
4. Based on these evaluations, select the better half of the sampling points.
5. Modify the sampling distribution according to the distribution found in the selected samples.
6. Return to step 2 and iterate until a desired stopping criterion is met

For an illustration of the adaptation process of the CMA-ES please see Figure 2.

3.1.3 Real Travel Times in Replanning

In the state of MATSIM-T we were using until now the replanning module was only in charge of finding a good time allocation for the activities given in the activity pattern. The travel times

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

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

To do so, we include a time of day dependent routing module in the replanning module. For every new candidate plan that is produced while the replanning module runs for a specific agent, the real travel time in the loaded road network is calculated for each of the trips based on their respective departure time.

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

3.2 Using the Optimal Replanning share

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

> Assume that in every iteration every agent has the opportunity to design a new plan for himself. Assume in addition that the network is very limited and that it consists only of two roads (A and B) leading from origin to destination. Let A be slightly shorter than B. In the first iteration, the Agents do not yet see yet any load on the network and consequently assume that road A is the better alternative to reach the destination. As all agents plan independently all come up with the same solution and as a result road A will be very congested during the first run of the microsimulation.

In the second iteration, all agents will perform a replanning. This time, the load on road A is visible to them. However, road B is still completely empty and now this alternative is the much better choice. Therefore, the agents decide to use road B in the next microsimulation run. This time, as all agents come to the same conclusion again, road B will be totally blocked while road A stays unnoticed and empty.

In the third iteration, we will face basically the same situation as in iteration one, leading to a cycle that will never end.

To avoid oscillating solutions the best approach probably would be to use only a very low replanning percentage. Although this is effective in producing a smooth replanning behavior, this leads to a poor learning speed of the overall system. One would expect that the system would learn more quickly—especially at the beginning—if the replanning percentage would be held at a reasonable high value.

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

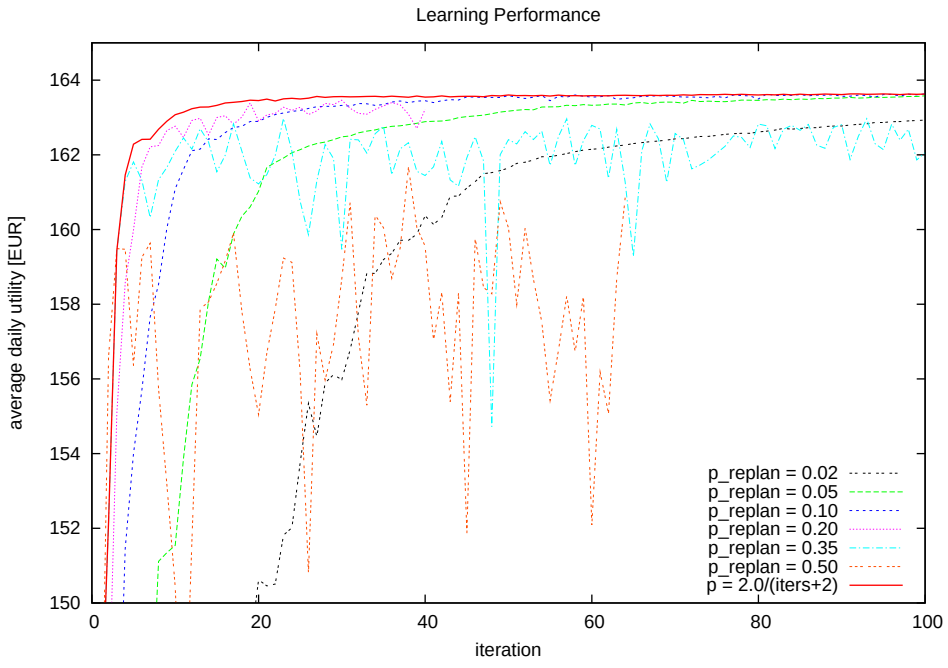


Figure 3: Test runs performed to find the optimal replanning share. The system was iterated using various parameters settings for p_{replan}, the replanning probability of an agent in each iteration. The tests were performed for fixed values in the range $2\% - 50\%$ and with a decreasing $p_{\text{replan}} = \min(35\%, \frac{2.0}{n+2})$, where n is the iteration number.

To answer this question we ran a couple of test runs of our system with different parameter settings for the replanning percentage in the range from $2\% - 50\%$. The results can be seen in

Figure 3. In good agreement with the reflections above we see that a replanning fraction of 50% produce a chaotic behavior of the simulation system. Below this value one can see that higher values lead to a faster learning process and lower values lead to less noise in the solution. For a constant replanning percentage, 5% and 10% seem to be reasonable values.

Judging from the shape of the convergence plots for constant replanning probabilities in Figure 3, we decided to use a decreasing replanning probability to get the best of both worlds: a quick learning process at early iterations and low noise and highly sophisticated results at the end. After some empirical testing we found that varying the replanning percentage according to following formula produced a significant improve in the learning performance of the system:

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

where p_{replan} is the replanning probability and n is the number of iteration. The comparison of the learning performances with constant and the variable replanning probabilities shows that the latter can boost the overall performance of the system by a factor of three or more.

3.3 Making iterations faster

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

3.3.1 The Time-Step-Based Microsimulation of Traffic Flow

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

and can be easily run in parallel. For a more detailed look at the microsimulation of traffic flow as we were using it so far see Cetin (2005).

3.3.2 The New Event-Driven Microsimulation of Traffic Flow

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

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

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

4. Test Scenario

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

The network we use has approximately 20k links and covers all of Switzerland. While it is possible for the agents to travel the whole network (e.g. an agent could theoretically decide to drive from one location in the canton of Zurich to another location in Zurich via Geneva, 300 kilometers away from Zurich) the part of the network that is used most of the time only comprises roughly 4000 links. In Figure 4 you can see the network in the city of Zurich, Switzerland.

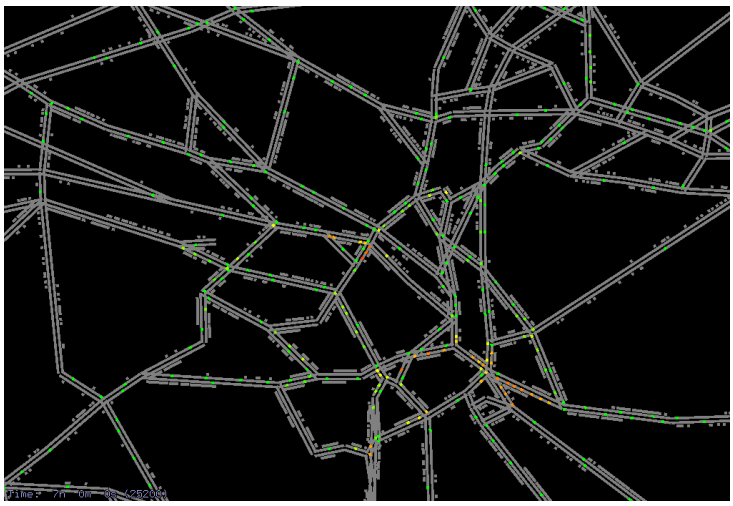


Figure 4: Extract from a scenario run. Shown: the city of Zurich, Switzerland. The colorful dots are cars driving through the network. Green, yellow, and red cars travel at free, half, and low speed, respectively.

During initial demand modeling, the activity chains as well as the locations for these activities are generated according to micro census data, a commuter matrix, and land use data. For details please refer to Balmer *et al.* (2006). The resulting initial demand for the learning loop of our agent-based system is shown in Figure 5. Note that it shows some observable properties of normal urban traffic like a morning and an evening rush hour but the the plans are very undifferentiated and certain properties are quite far from reality such as the evening peek that starts already at 14:00.

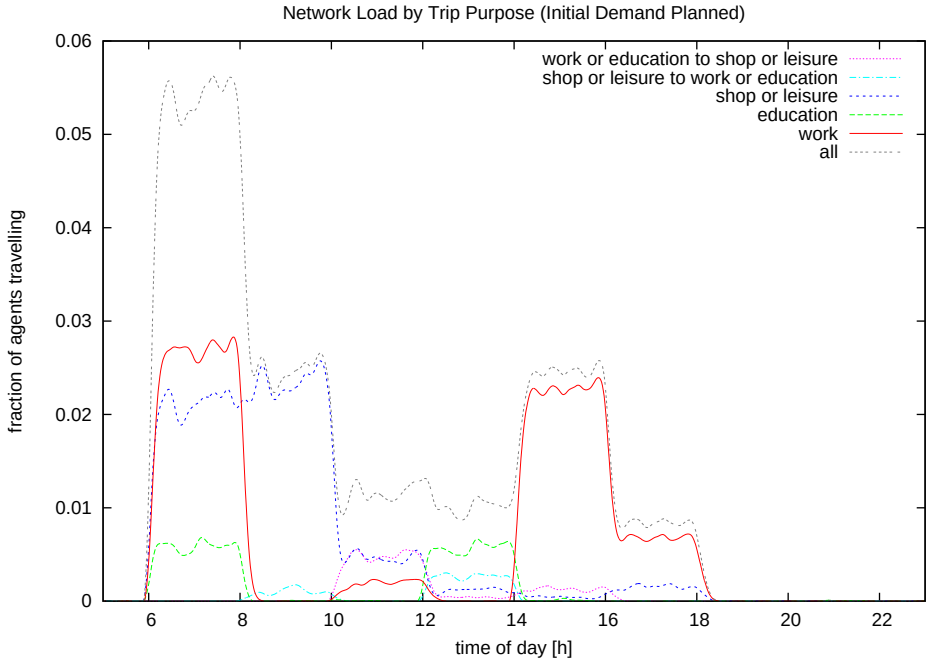


Figure 5: The network load according to the initial daily plans of the agents before iteration 0. Note that the demand cannot be executed in this way. The plans of the agents assume an empty road network and corresponding travel times. The real execution leads to a very congested network and therefore to a big discrepancy between the planned daily plans and the executed ones.

The average number of trips that is carried out by each agent is 2.24 and the average trip length in number of links traveled is 6.94 after convergence to the user equilibrium.

5. Results

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

5.1 Comparison of Learning Performance Based on Average Utility of Plans

To measure how well the agents in our simulation have learned to perform their activities and to adapt to the system under load we use the average utility of their plans as they were executed

in the microsimulation module. In the following plots higher values usually mean a better adaptation to the problem at hand. However, higher average values do not always depict a better overall performance. One has to keep in mind that we are looking for a user equilibrium while the average score would be at a maximum in the system optimum, these two not necessarily being the same. Unfortunately, it is not easily possible to find out if the agents are in a (stable) user equilibrium. For this reason we nevertheless use the average score to judge the performance of our simulation system.

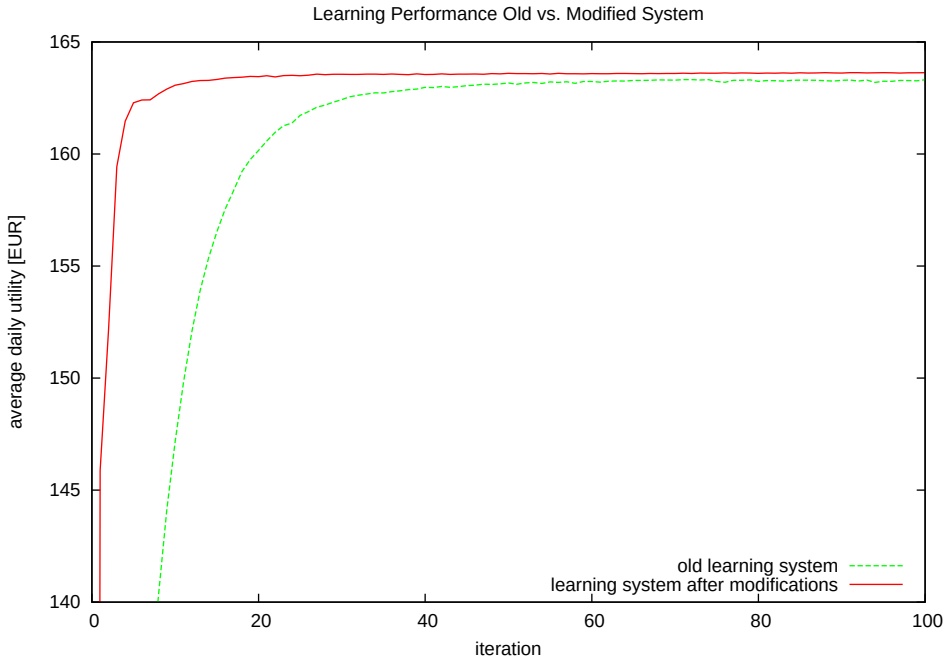


Figure 6: The simulation system after modifications learns substantially faster than in the original state. Also a higher maximum average utility is reached.

In Figure 6 a comparison of our simulation system before any changes and the current state can be seen. The system in the new state learns substantially faster than the original system. The modified system reaches the stable plateau close to the maximum after approximately 20 iterations. The original system shows a local maximum at about 70 iterations but continues to rise after iteration 100 and reaches a similar average utility after approximately 900 iterations (not shown). If we take iteration 70 as final result of the original setup this means that our new system is a factor of 3.5 faster in terms of iterations.

5.2 Comparison Of Results Using Network Loads

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

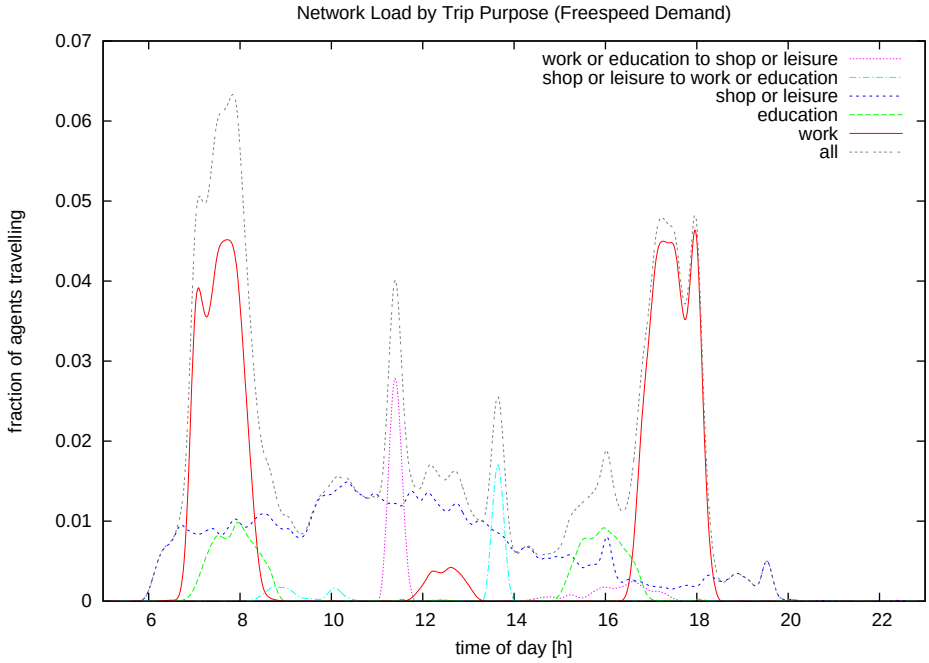


Figure 7: The demand as it is planned by the agents assuming free speed travel times on all links. The morning and evening peeks present are very pointed. Note that shop and leisure activities happen throughout the day and that they do not interact with the rush hours. As were assuming free speed there is no need to avoid heavy traffic during morning. It is clear that this demand cannot be executed in this way—the network would become totally overloaded.

In Figure 7 we see a fictitious network load as it is planned by the agents if they assume infinite network capacities and free speed travel times accordingly. High values in this plot mean that many agents are traveling at the same time. This plot can be used as an indication of how the agents would plan their days if there was a guarantee of no congestion at any time. Note the pointed morning and evening peeks and the relatively flat distribution of shop and leisure trips throughout the day. It is interesting to compare this “ideal” demand to the real executed demand shown in Figure 8 as it was found using our new learning system after 20 iterations. It can be seen that the morning and evening peeks are significantly broader (the morning peek starts earlier and the evening peak ends later) and that people on shopping and leisure trips systematically avoid these peek hours for their travel. We conclude that our learning system is able to reproduce a typical behavior of people in daily life: They avoid peek hours for travel to and from activities that are not bound to a specific time.

In Figure 9 we show the same plot as above for the original, unmodified learning system after 70 iterations. The peak hours look very similar to Figure 8 but no interaction effects between shopping and leisure activities and the peak hours can be identified. This shows that the the original system needs significantly more time to reach a certain quality of solution. After further investigation we found that in iteration 200 the old system already has developed the avoidance effects partially, but that it takes roughly 400 iterations before a similar quality can be observed.

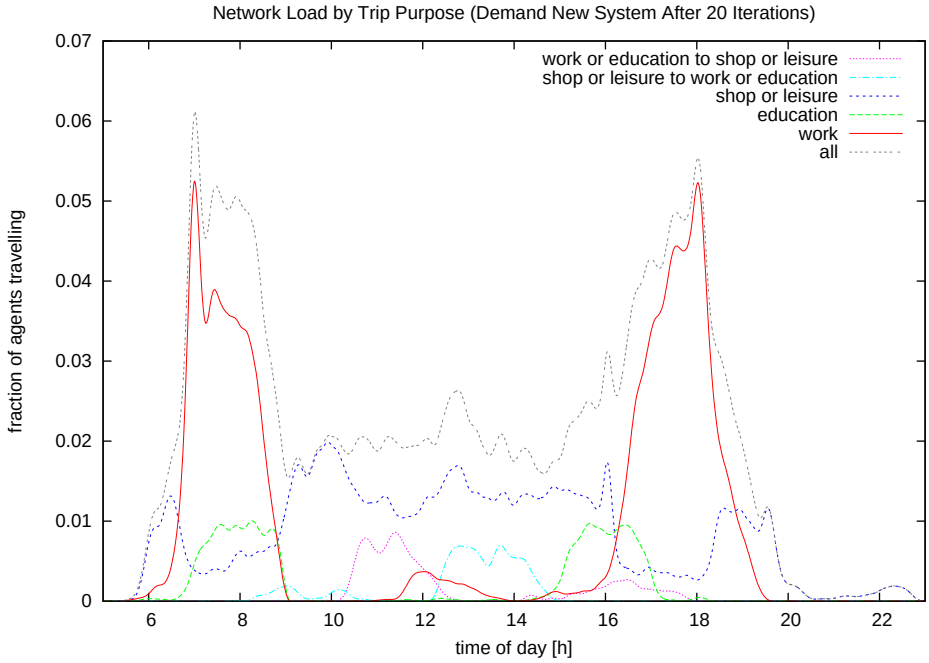


Figure 8: Demand after 20 iterations of learning of our new simulation system. The agents have learned to avoid the morning and evening peeks if possible. These peeks are still very dominant but significantly broader than in the ”free speed” demand. Note how shopping and leisure trips are carried out during off-peek hours to avoid congested roads.

The new system therefore needs 20 times less iterations to reach the final quality of solution that the old system.

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

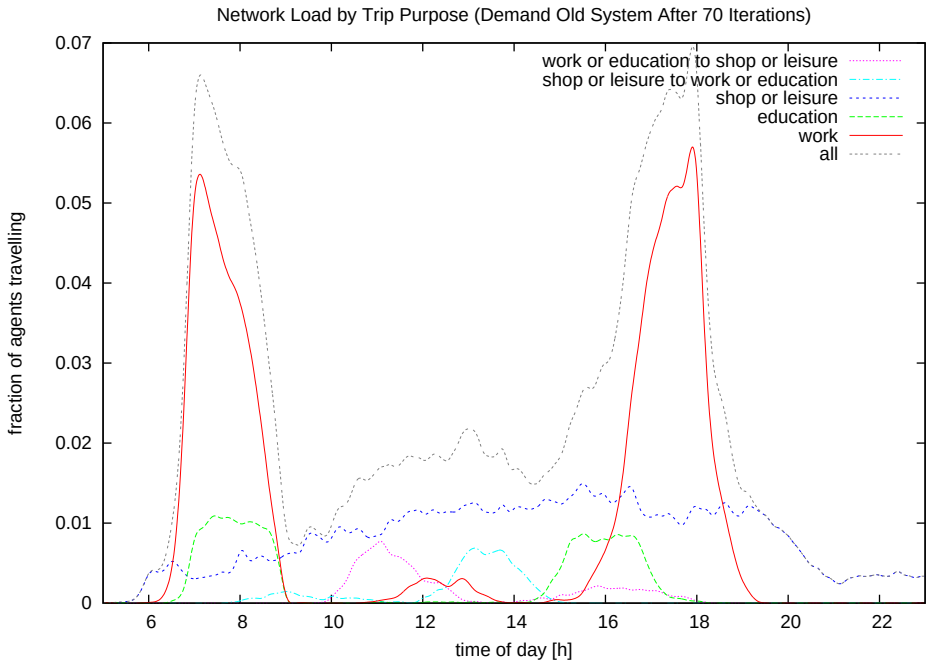


Figure 9: Demand after 70 iterations of learning of the original learning system. While the morning and evening peaks have developed, shopping and leisure trips do no yet avoid the peak hours.

6. Conclusion and Outlook

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

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

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

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

References

Balmer, M., K. W. Axhausen and K. Nagel (2006) An agent-based demand-modeling framework for large scale micro-simulations, in *TRB 85th Annual Meeting Compendium of Papers CD-ROM*, Transportation Research Board, Washington, D.C.

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

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

Frick, M. and K. W. Axhausen (2004) Generating synthetic populations using IPF and monte carlo techniques: Some new results, in *The 4th Swiss Transport Research Conference (STRC)*, Monte Verità, Ascona, `http://www.strc.ch/2004.html`.

Hansen, N. and S. Kern (2004) Evaluating the CMA evolution strategy on multimodal test functions, in *The Eighth International Conference on Parallel Problem Solving from Nature*.

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

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