Transportation (2005) 32: 369–397

Generating complete all-day activity plans with genetic algorithms

DAVID CHARYPAR[1] & KAI NAGEL[2,*]
[1]*Department of Computer Science, ETH Zürich, Switzerland;* [2]*Institute for Land and Sea Transport Systems, TU Berlin, Germany*
*(*Author for correspondence, E-mail: nagel@kainagel.org)*

Key words: activity generation, genetic algorithms, location choice, multi-agent traffic simulation, utility functions

Abstract. Activity-based demand generation contructs complete all-day activity plans for each member of a population, and derives transportation demand from the fact that consecutive activities at different locations need to be connected by travel. Besides many other advantages, activity-based demand generation also fits well into the paradigm of multi-agent simulation, where each traveler is kept as an individual throughout the whole modeling process. In this paper, we present a new approach to the problem, which uses genetic algorithms (GA). Our GA keeps, for each member of the population, several instances of possible all-day activity plans in memory. Those plans are modified by mutation and crossover, while 'bad' instances are eventually discarded. Any GA needs a fitness function to evaluate the performance of each instance. For all-day activity plans, it makes sense to use a utility function to obtain such a fitness. In consequence, a significant part of the paper is spent discussing such a utility function. In addition, the paper shows the performance of the algorithm to a few selected problems, including very busy and rather non-busy days.

1. Introduction

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

– The process starts by generating a *synthetic population* from census data (e.g. Beckman et al. 1996).
– Next, for each synthetic person of the synthetic population a *plan* is generated. Plans consist of activity patterns, activity locations, activity

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

For this approach, many collaborating modules need to be designed, implemented, and tested. An important part of those modules concerns activity generation: for each synthetic individual, a sequence of activities is generated, including activity location and activity times. Activity-based demand generation is a very active field of research; see, e.g., the proceedings of two recent conferences (IATBR'03, 2003; EIRASS'04, 2004). The mainstay of activity-based demand generation are random utility models (RUMs) (Ben-Akiva & Lerman 1985; Bowman et al. 1999; Pendyala, accessed 2004; Bhat et al. forthcoming). RUMs, however, arguably have the disadvantage that they are behaviorally not very realistic. In consequence, alternatives are also investigated, such as behaviorally or rule-based approaches (e.g. Arentze et al. 2000; Miller & Roorda 2003).

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

The paper starts with a more precise problem description (Section 2), followed by a short review of previous work in the area of activity generation (Section 3). Section 4 then discusses a concept of how GA could be used to generate daily activities; Section 5 contains details about our specific computational implementation. GAs work by maintaining a population of

solutions; they improve the best known solution by mutating and combining members of that population. In order for this to work, individuals need to be given scores. This is normally called a fitness function; in social science research, it is plausible to use utility functions instead. In consequence, Section 6 describes the requirements that a utility function for the GA approach needs to fulfill, and which particular utility function we selected. However, *any* function that gives scores to activity chains will work. Section 7 then contains tests and results for several illustrative examples. The paper is concluded by an outlook on future work (Section 8) and a summary (Section 9).

2. Problem description

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

Activity planning divides into three subproblems:

- The first subproblem is to select activities to be executed and to decide in which order they should be executed. We refer to this subproblem as *activity pattern generation*.
- The second subproblem is to find the places where the activities are going to be executed. We call this *location selection*. The location selection has to fulfil a number of constraints in order to be meaningful. For instance children should be fetched from school at the same place where they were dropped off before.
- The third subproblem is to decide when the activities have to be executed and for how long. We call this *time allocation*. Once again, time allocation is subject to several restrictions. The simplest one is that the execution times should be ordered in correspondence with the activity pattern.

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

To make the problem even worse, it would be desirable to extend the length of the activity plan to a week or even a month because day plans are

not independent in general as some activities do not have to be executed every day. For instance you do not have to go shopping every day, but you also cannot omit shopping for a longer period.

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

3. Related work

There are many models tackling the same problem. One can, may be, differentiate the following directions:

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

Discrete choice models have a similar conceptual approach as our model in that they make choices based on utilities. The two main differences are that, at least conceptually, discrete choice models enumerate *all* possible alternatives, and that they do not choose the option with the best utility but they choose between options with probabilities which are related to utilities. The first aspect means that a huge number

of options needs to be considered; the second (behaviorally somewhat justified) aspect means that efficient search methods such as branch-and-bound cannot be used because even 'bad' branches of the search tree have a probability that they will be selected.

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

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

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

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

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

4. Idea: Genetic algorithms

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

The idea for this paper is to use a GA to find good all-day activity plans. GAs have been used for many problems with huge search spaces; a suggestion to use them in the context of transport/land-use research is by Abraham and Hunt (2002). GAs maintain a population of solution

instances during the search process, and search progress is made by mutation, crossover, and selection, as explained below. In our case, the population of solution instances consists of many possible day plans for a single given traveler. The quality of such a day plan is rated by a fitness function, which uses informations and restrictions known about the activities, and estimates how well the day plan meets them.

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

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

5. Implementation

As mentioned above, crossover, mutation, and selection are vital components of each GA. Before these operators can be defined, it is necessary to come up with a way of representing a solution instance in the computer. This way of representation is referred to as *encoding*. Encoding is of prime importance for the definition of crossover and mutation and it has a large influence on the potential performance of a GA. For our problem of activity planning, we used a combination of binary encoding, permutation encoding and value encoding in the following way (Figure 1):

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

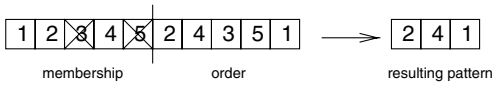


Figure 1. Encoding of the activity pattern '241'.

- A second array stores the order of the activities (permutation encoding). It holds always a full set of activities, regardless of which activities are actually member of the day plan and which not. When evaluating the day plan, the positions with disabled activities are ignored.
- Our day plans also include information about the location choice. In our model, we assume that there exists a facility that is assigned to each activity. This facility can be thought of as a type of building or place that is needed in order to perform the activity. For instance, for the activity 'go shopping' one needs the facility 'shop'. We also assume that for each facility there exist multiple locations as for example there are multiple shops in a city where shopping can be done. The concept of facilities makes it possible to have locations of activities that depend on each other. For instance, this is needed to take care of the fact that one has to fetch one's children at the same school as one has dropped them off before. The day plan has to store the selected location for each facility. This information is held in a third array storing the ID of the location for each facility (value encoding).
- The activity durations are stored in a fourth array which stores floating point numbers (once more value encoding); in addition, there is an entry, which contains the starting time of the day plan. Activity starting times of the allocated time slots are a result of adding up all previous durations and travel times (see below).

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

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

- First, the array which stores the membership information is processed using uniform crossover. That is, each membership bit is copied randomly from either one of the parents.
- Second, we crossover the arrays that store the order information (Figure 2). Before we start, we decide randomly which parent should precede (i.e. which parent's activity should come first) in case of a collision. For each activity ('1' to '5' in Figure 2), we randomly select a parent and put the activity at the same position in the offspring as it was in the selected parent (see '(*)' in Figure 2). If there are collisions (i.e. two activities in the same cell), then their sequence is decided first (see '(**)'), and then the resulting new sequence is copied into its correct

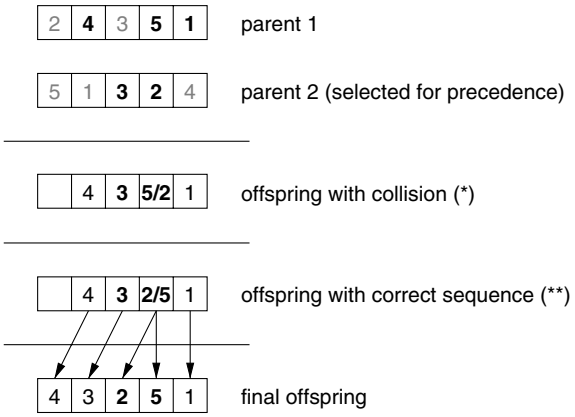


Figure 2. Illustration of crossover operator.

location. This, somewhat involved algorithm was developed since it does not depend on the sequence in which it goes through the activities, and therefore it does not introduce a bias caused by the ordering in which the activities are processed in this sub-step.

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

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

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

shows a considerable tendency to keep stuck in local optima when using single point crossover.

The *mutation operator* is split into four parts: first, for each activity, the membership information is flipped with a probability p_{mut}. Second, the order of the activities is permuted: with a probability p_{mut} two activities, both chosen at random, are exchanged. This is done n times, where n is the total number of activities. Using the same idea as in the crossover operator, the activities are exchanged ignoring the membership information. Third, the duration of each activity is changed by multiplying it with a factor $f = e^X$, where X is drawn uniformly from the interval $[-p_{\text{mut}}/2, p_{\text{mut}}/2]$. Fourth, a new start time of the activity plan is calculated by adding a random time uniformly drawn from $[-p_{\text{mut}} \cdot 12h, p_{\text{mut}} \cdot 12h]$.

6. Utility function

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

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

$$F = \sum_{i=1}^{n} U_{\text{act}}(\text{type}_i, \text{start}_i, \text{dur}_i) + \sum_{i=2}^{n} U_{\text{trav}}(\text{loc}_{i-1}, \text{loc}_i) \tag{1}$$

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

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

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

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

$$U_{\text{actl},i} = U_{\text{dur},i} + U_{\text{wait},i} + U_{\text{late.ar},i} + U_{\text{early.dp},i} + U_{\text{short.dur},i}. \tag{2}$$

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

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

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

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

6.1. *Utilities for performing activities*

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

$$U_{\text{dur}}(t_{\text{dur}}) = \beta_{\text{dur}} \cdot t^* \cdot \ln\left(\frac{t_{\text{dur}}}{t_0}\right). \tag{3}$$

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

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

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

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

$$\frac{\partial U}{\partial t_{\text{dur}}}(t_{\text{dur}}) = \frac{\beta_{\text{dur}} \cdot t^*}{t_{\text{dur}}} \tag{4}$$

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

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

- It determines the minimum duration of an activity. If the duration falls below this value, then it is more beneficial to drop the activity and do nothing instead.
- It determines the priority of an activity: The marginal utility at $t_{\text{dur}} = t_0$ is

$$\frac{\partial U}{\partial t_{\text{dur}}}(t_0) = \frac{\beta_{\text{dur}} \cdot t^*}{t_0}. \tag{5}$$

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

In our case, the specific form of

$$\alpha = e^{-200/(t^* \cdot p \cdot \beta_{\text{dur}})} \tag{6}$$

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

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

6.2. *Penalties*

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

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

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

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

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

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

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

At this point, one could discuss plausible relations between the different β, for example using the typical Vickrey scenario values of $-6/h$, $-12/h$, and $-18/h$ for β_{wait}, β_{trav}, and $\beta_{\text{late.ar}}$. However, as will be discussed in Section 8, in our framework there are interactions between the marginal utility of doing activities and the *effective* marginal utility of the penalty terms, so that the effective marginal utilities that guide behavior are more complicated than one would assume by the above numbers. This is caused by the fact that some of the penalized items, such as waiting and traveling, also incur the *additional* penalty of not being able to earn positive utility from doing an activity at the same time (opportunity cost). The other penalized items – arriving late, leaving early, or staying for too short – do not have this property. Therefore, the values used in the tests will just be stated in Section 6.4 and be used without further comment except for the discussion in Section 8. Once more, please note that this paper describes a prototype implementation rather than an operational model.

6.3. *Opening times and similar constraints*

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

One possibility would be to assign a very low utility to those activities, e.g. minus infinity. This policy would be very efficient in avoiding invalid day plans. But it would not make very much sense, for the following reason:

assume that you have to compare two time allocations for the activity 'shop'. In the first scenario, you go shopping at 7:59 am and shop for 2 h. In the second scenario, you go shopping at 8:00 am and shop also for 2 hours. The shop opens at 8 am. The first time allocation is invalid because you try to shop while the shop is closed. So the utility of the first scenario would be very low. The second scenario, however, has a very high utility. With this policy, the minimal change of one minute in time allocation would produce a huge change in utility which is certainly not realistic. In reality, we would have waited one minute in front of the shop. That means that one should set parameters such that both utilities are almost the same.

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

- At the beginning of the allocated time slot, we assume that the agent travels from the location of the last activity to the location of the new activity. The time spent for traveling yields a (dis)utility according to the section about travel costs.
- From the moment of arrival at the activity location until the end of the time slot, as much time as possible is spent actually performing the activity.
- If for some reason, for instance because of opening hours, it is not possible to use all this time for performing the activity, the remaining time is spent waiting. Waiting yields a negative utility of $\beta_{\text{wait}} \cdot t_{\text{wait}}$.
- Since we use a logarithmic function for the utility of duration, it is possible that this utility becomes negative. If this is the case, and if it is more efficient to spend the whole time waiting,[2] the activity is not executed at all. It may sound weird to travel to an activity that is not executed. However, it is important to note that we need to calculate meaningful utilities for no matter which day plans the GA generates, because this is the material the GA works with. Since traveling to an activity location without executing the activity does not make sense, the GA will eventually find a better solution, such as either completely dropping the activity, or allocating sufficient time for it.

6.4. *Summary of parameters*

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

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

These parameters were selected in order to match the typical Vickrey scenario values of -6, -12, and -18. Upon further reflection, one does, however, recognize that the above values in conjunction with our approach do not do this, since when traveling or waiting, one incurs the *additional* penalty of time that one could spend doing an activity, at the 'cost' of -20/h (opportunity cost). Section 7 discusses better values.

6.5. *Examples of utility landscapes*

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

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

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

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

We first show the typical utility landscape of an activity chain with four activities *home*, *work*, leisure and *home*. The t^* for *work*, *leisure* and *home* had been set to 8, 2 and 4 hours, respectively. There were no additional constraints that had to be fulfilled except that the total length of the plan was fixed to 16 hours (Figure 3).

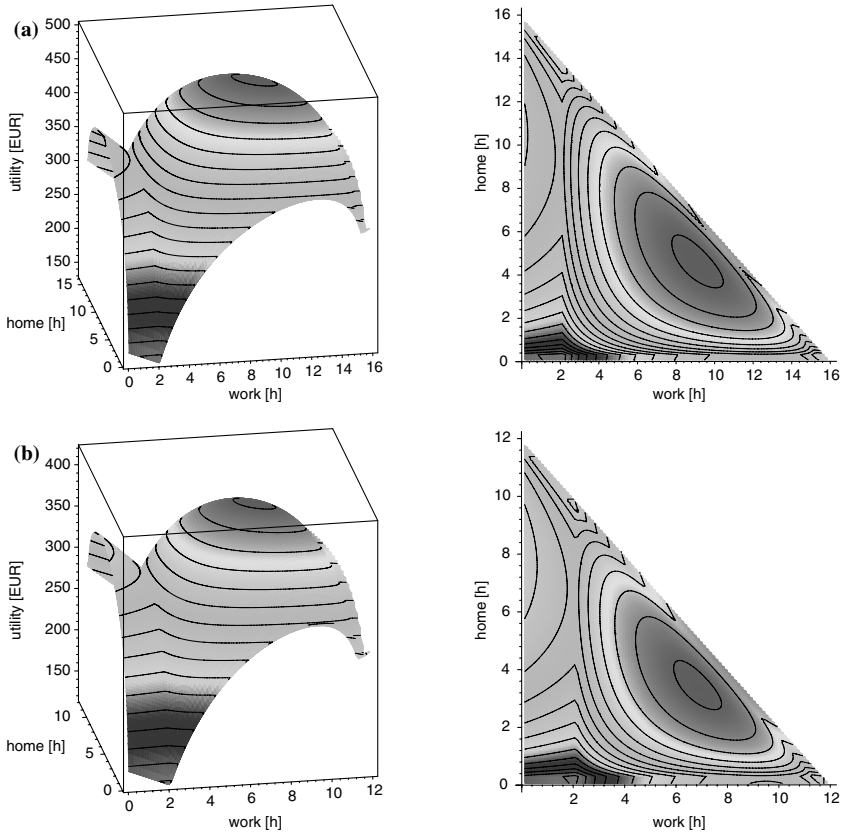


Figure 3. Utility of activity pattern *home-work-leisure-home*. t^* of work, leisure and home set to 8, 2 and 4 hours, respectively. (a) Total time budget of 16 hours. (b) Total time budget of 12 hours.

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

In order to show the effect of the time budget on the time allocation, we show another plot of the same activity pattern but this time with a desired plan length of 12 hours (Figure 3). The optimum is shifted towards shorter times and the utility of the optimum is roughly 100 lower. The utility

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

In Figure 4 we see that now there exist three local optima. The global optimum (top left) corresponds to showing up in the shop at 9 in the morning and staying there until 17 in the evening. The lunch break is spent waiting. The two local optima in the morning and the afternoon are much more meaningful. The early local optimum corresponds to coming at 9 am when the shop opens and leaving the shop at 11 am when it closes. The late local optimum corresponds to coming at 14 when the shop opens and leaving at 17 when it closes.

The structure of the fitness (utility) landscape becomes even more complex when we consider an activity *chain* that includes a complex activity with opening hours. Figure 5 shows the utility of the activity pattern *work-shop-work*. t^* was set to 8 hours for the total working duration and 2 hours for the shopping activity. The overall plan was set to start at 8 and last until 17. One sees that good plans set the shopping time to 1.5 hours; the overall work time will then be set to 7.5 hours.

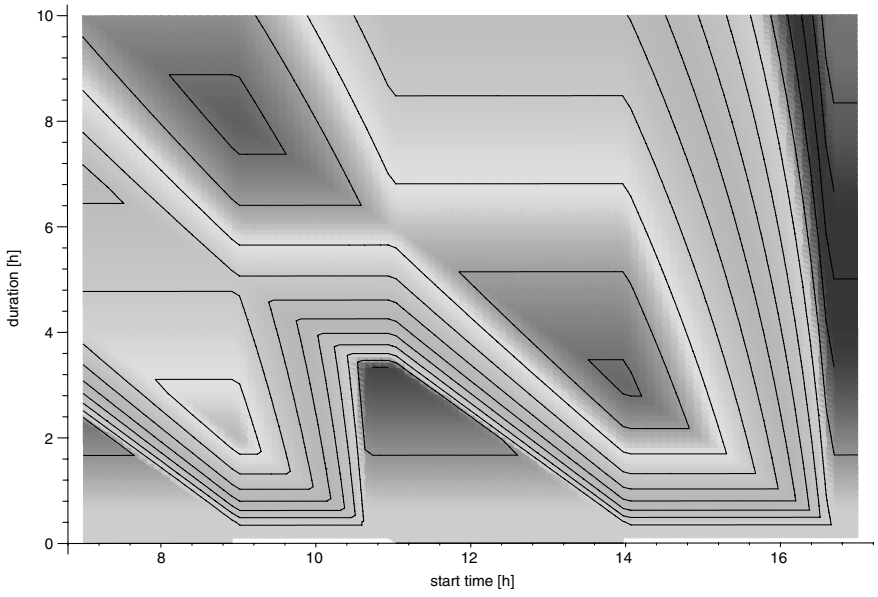


Figure 4. Utility of activity *shop*. Shop open 9–11, 14–17. There is no constraint on the total length of the partial plan.

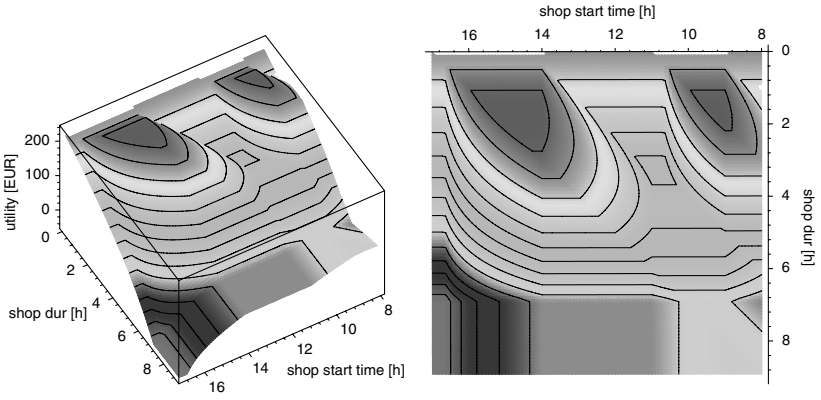


Figure 5. Utility of activity pattern *work-shop-work*. The shop is open 9–11 and 14–17. For the calculation of the utility of work only the total working duration is regarded. That is, the sum of working duration before shopping and after shopping is first calculated and then the utility for work is calculated. Note also that the axes in this plot are reversed for visibility reasons.

The utility for this activity pattern shows two equivalent global optima. They correspond to go shopping in the morning for almost 2 hours and to go shopping in the afternoon. The optimum in the afternoon is wider, because the shop is open for 3 hours in the afternoon.

7. Tests and results for complete day-plans

In this section, we want to show the general ability of our model to generate complete day plans. For that purpose, we designed a simple city. In our city there exist a number of different facilities including apartments, working places, shops, kindergartens and recreational areas. For each of these facilities, there exist three different locations between which an agent can choose. The different locations of the facilities are summarized in a map. See, e.g., Figure 7.

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

- Run the algorithm while long-term locations (e.g. home, work, kindergarten) are fixed in order to determine the short-term flexible parts of a daily schedule.
- Run the algorithm, say, for a possible new home location in order to test if an agent could improve its daily utility function by moving to this

location (Abraham & Hunt 2002). In this situation, the algorithm would model the 'what-if's' of humans when they consider alternative options without actually executing them.

– Clearly, any intermediate level is possible as well, for example when home and work are kept fixed but a kindergarten location is searched for.

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

We want to test our algorithm for different kinds of agents with different lists of activities that they would like to accomplish. For this purpose, we defined three different scenarios. Each scenario is defined by a set of activities that are available. The scenario 'full10' is the one with the largest set and consist of 10 activities which are listed in Table 1. The other two scenarios use a subset of these activities.

All facilities needed by the activities except for the facility 'home' are not open during the whole day. The opening times used for our investigations are summarized in Table 2.

The three scenarios are defined as follows:

– As mentioned above, the 'full10' scenario defines a scenario with set of 10 activities. This scenario consists of the activities 'sleep', 'breakfast', 'lunch', 'dinner', 'early work', 'late work', 'bring children to kindergarten', 'fetch children from kindergarten', 'shop', and 'leisure'. The

Table 1. Activities of our test case.

Name	Priority	t^*[1]	$t_{\text{latest.ar}}$[2]	$t_{\text{earliest.dp}}$[3]	$t_{\text{short.dur}}$[3]	Facility
Sleep	1	8.0	25.0	29.0	6.0	Home
Early work	1	4.0	9.0	11.0	3.5	Work
Late work	1	4.0	14.0	15.0	3.5	Work
Breakfast	3	0.5	10.0		0.25	Home
Lunch	2	1.25	14.0	12.0	0.75	Work/Home[4]
Dinner	2	2.0	21.0	18.0	0.75	Home
Bring to kindergarten	1	0.25	9.0	8.5	0.25	Kindergarten
Fetch from kindergarten	1	0.25	16.0	15.5	0.25	Kindergarten
Shopping	3	2.0			0.5	Shop
Leisure	3	2.0			1.0	Leisure

[1] t_0 (not shown in table) is derived from the priority and t^*. See Section 6.1 and Eq. (6).
[2] Latest starting time. If the activity starts later a penalty is applied. See Section 6.2.
[3] Both $t_{\text{earliest.dp}}$ and $t_{\text{short.dur}}$ address the problem of not executing an activity long enough. If the activity is ended before $t_{\text{earliest.dp}}$ a penalty applies. The same penalty applies if the duration of the activity is shorter than $t_{\text{short.dur}}$. See Section 6.2.
[4] Depending on the scenario, the facility of the activity 'lunch' was either work or home. See description of the scenarios.

Table 2. Opening times of the facilities. An activity can only be carried out when the required facility is open.

Facility name	Opening times
Home	00:00–24:00
Work	06:00–20:00
Kindergarten	08:30–9:00 15:30–16:00
Shop	09:00–19:00
Leisure	14:00–24:00

Note: Facility "Kindergarten" has two opening times. The first one defines when children can be dropped off and the second when they can be picked up.

purpose of this scenario is to show the performance of our algorithm if there are very many activities to perform.

– In the 'houseman' scenario, the agent has only 8 of the 10 activities of the 'full10' scenario to choose from, leaving out both work activities. That is, the remaining activities are: 'sleep', 'breakfast', 'lunch', 'dinner', 'bring children to kindergarten', 'fetch children from kindergarten', 'shop' and 'leisure'. Since in this scenario the agent does not have a work location, we changed the facility for eating lunch from 'work' to 'home'. This scenario simulates a rather dense day plan of a non-working person.
– In the 'pensioner' scenario, the set of activities consists only of 5 activities; these are 'sleep', 'lunch', 'dinner', 'shopping' and 'leisure'. As in the 'houseman' scenario, the facility for having lunch was changed to 'home'. With this scenario we want to show how our model deals with day plans with a large freedom in time allocation.

In all these scenarios, the set of activities is endogenous to the model. This is discussed in Section. 8.

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

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

generations is limited to 200,000. Here, the execution takes between 3 and 5 seconds.

The quality test for the 'full10' scenario was run with 5 different initializations. At the end of each run, the day plan with the highest utility value was always identical in terms of generated pattern and location selection. Only the time allocations differed, but the differences were within very few minutes. In Figure 6 we show a typical convergence graph for the quality tests. The best day plan found for the 'full10' scenario in the quality test is shown in Figure 6. Note that the children are not dropped off at home before continuing with the day plan; especially 'leisure' is performed together with the children. This can be observed in almost all generated day plans. There is nothing that would force the agents to drop children off before starting with the next activity. However, if desired, this could be easily modified in the utility function.

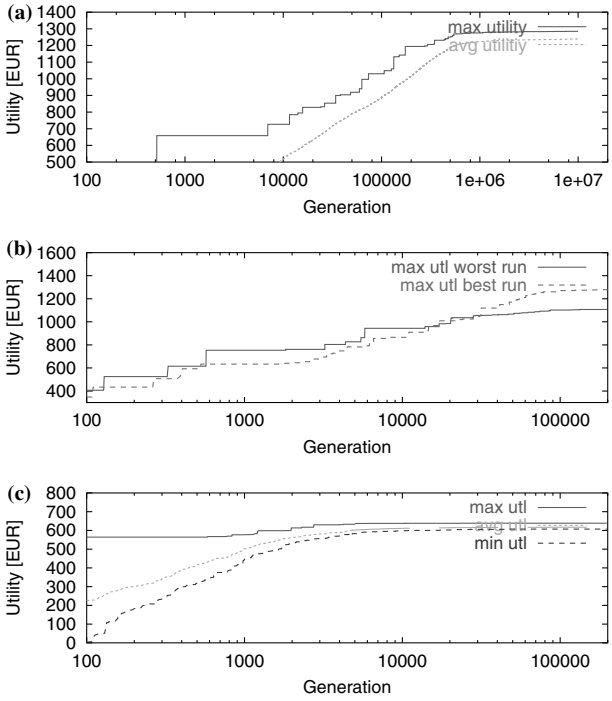
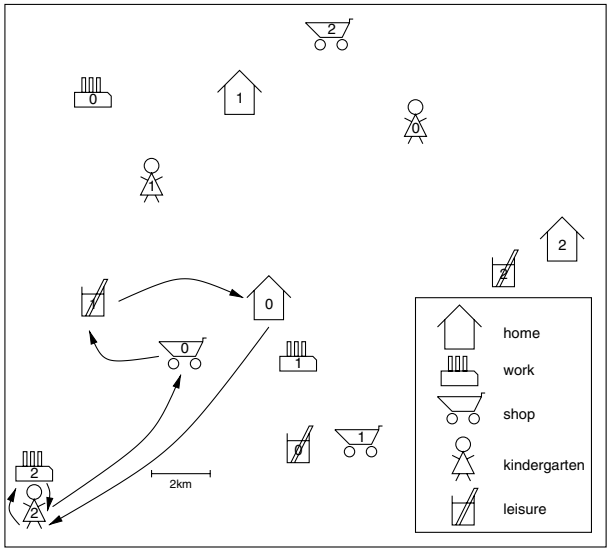


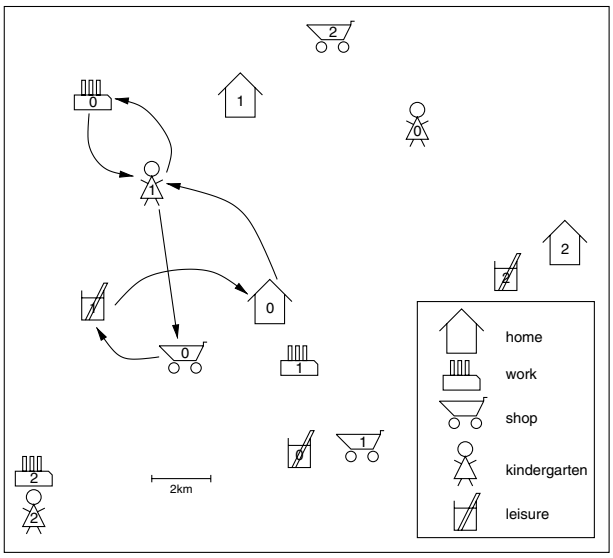
Figure 6. Convergence graphs. (a) 'Full10' scenario with 10 million generations in order to get the best possible quality of the resulting day plan. The graph shows the maximal and average utility of the population of a typical long run, (b) 'Full10' scenario with only 200,000 generations to test the performance of the algorithm if there is only limited time for finding a good day plan. The graph shows the maximum utility of the population for the best and the worst short run and (c) 'Pensioner' scenario for a typical short run. In this test, the goal was to find a good day plan within limited time.

In the five short runs for investigation of speed and usability for the scenario 'full10' three times a solution was found that is identical to the one shown in Figure 7 in terms of generated pattern and selected locations. Only the time allocations were not as sophisticated. The total utility varied between 1277.54 and 1278.84. In the two remaining runs, two different solutions were found. The solution shown in Figure 8 differs only in terms of location selection and time allocation while the solution shown in Figure 9 differs also in the generated pattern. Note that in this day plan the activity 'bring children to kindergarten' is left out. In Figure 6 we compare the convergence of the best and the worst short run.



Activity Name	Travel Time	Execution Time	Location
Breakfast		06:56–07:26	home0
Bring children[1]	07:26–08:30	08:30–08:40	kiga[3] 2
Early work	08:40–08:46	08:46–11:52	work 2
Lunch		11:52–12:40	work 2
Late work		12:40–15:40	work 2
Fetch children[2]	15:40–15:46	15:46–16:00	kiga[3] 2
Shop	16:01–16:43	16:43–18:26	shop 0
Leisure	18:26–18:48	18:48–20:27	leisure1
Dinner	20:27–21:03	21:03–23:02	home0
Sleep		23:02–06:56	home0

Figure 7. Best day plan for the 'full10' scenario after 10 million generations. The total utility is 1284.93. The map graphically summarizes the day plan.

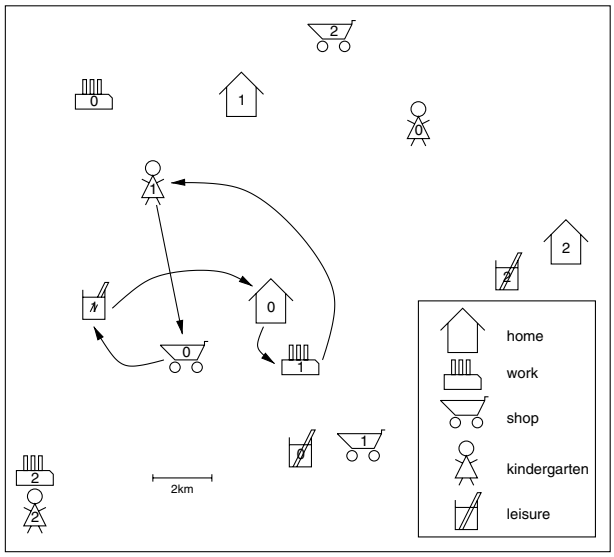


Activity Name	Travel Time	Execution Time	Location
Breakfast		07:27–07:56	home 0
Bring children	07:56–08:30	08:30–08:42	**kiga 1**
Early work	08:42–09:03	09:03–11:54	**work 0**
Lunch		11:54–12:49	**work 0**
Late work		11:54–12:49	**work 0**
Fetch children	15:18–15:39	15:39–15:57	**kiga 1**
Shop	15:57–16:33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

Figure 8. First alternative day plan for the 'full10' scenario after 200,000 generations. The total utility is 1276.46.

In all runs for the 'pensioner' scenario (five long runs and five short runs) the same day plan was found with respect to the generated activity pattern and the selected locations. The time allocations to the different activities was also very similar. The similarity of the day plans can also be seen when looking at the total utility which was always between 638.483 and 638.514 – a very narrow interval. The only difference between the plans was their starting time which varied in the range from 10:33 am to 11:45 am. This is due to the lack of constraints for the activities. That is, it does not matter at which time inside the range the day plan starts.

It seems that our algorithm has no problems to find a good solution for the 'pensioner' scenario. In order to test this assumption, we show the

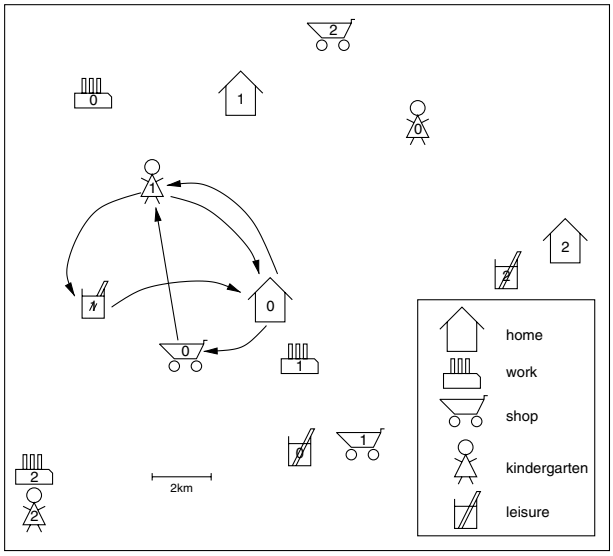


Activity Name	Travel Time	Execution Time	Location
Breakfast		06:14–06:46	home 0
Early work	06:46–06:59	06:59–10:56	work 1
Lunch		10:56–11:59	work 1
Late work		11:59–15:04	work 1
Fetch children	15:04–15:51	15:51–16:00	kiga 1
Shop	16:00–16:37	16:37–18:26	shop 0
Leisure	18:26–18:47	18:47–20:27	leisure 1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

Figure 9. Second alternative day plan for the 'full10' scenario after 200,000 generations. The total utility is 1107.89. Note, the activity 'bring children to kindergarten' is left out.

convergence graph of one of the usability runs in Figure 6. One can see that the algorithm converges already very early. In fact, a solution identical to the one found in the end in terms of generated pattern and selected locations is already found after 30,000 generations – which is only 15% of total number of generations.

In all five long runs for the 'houseman' scenario and in four of the five short runs the same day plan was found with respect to the generated activity pattern and the selected locations. The time allocations were also very similar, they all lay within 10 minutes. The resulting utilities vary between 1040.51 and 1043.04. We show the best day plan found in Figure 10.



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

Figure 10. Best day plan found for the 'houseman' scenario. The total utility is 1043.04. Note that the agent fetches the children and performs activity 'leisure' with them. The reason why it does not drop them off at home before is that there is no constraint forcing it to do so and the presented day plan minimizes travel time.

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

The resulting day plans for all three scenarios are plausible. All aspects of activity planning – activity pattern generation, location selection and time allocation – are taken care of. The convergence of the genetic algorithm seems to be very stable, as for each scenario at the end of almost all test runs the same good solution is found. It seems, that our tests have not yet pushed our algorithm to its limits, it would be interesting to see how it performs on generating complete week plans.

8. Discussion and further work

The values for the marginal utilities (Section 6.4) were selected to match typical values of the Vickrey scenario (e.g. Arnott et al. 1993). It turns out, however, that in our framework the *effective* penalties of traveling or waiting are not the values of Section 6.4, but they are those values *plus* the utility lost by not doing any activity, i.e. −20/h (opportunity cost). This means that the *effective* marginal disutilities of traveling and waiting are −32/h and −26/h, respectively. The correct values of our system to obtain the typical Vickrey penalties of −6, −12 and −18 for waiting, traveling, and being late are thus

Marginal utility of any activity: $\beta_{\text{dur}} = 6\,€/h.$
Marginal utility of travel time: $\beta_{\text{trav}} = -6\,€/h.$
Marginal utility of waiting: $\beta_{\text{wait}} = 0\,€/h.$
Marginal utility of coming late: $\beta_{\text{late.ar}} = -18\,€/h.$
Marginal utility of leaving early: $\beta_{\text{early.dp}} = -18\,€/h.$

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

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

In all scenarios of Section 7, the set of activities is endogenous to the model. Such sets could for example come from separate models of activity repertoires. However, our algorithm is in principle capable to drop low priority activities, although that was not systematically investigated. Therefore, one could imagine to start from a maximum set of activities, possibly with the desire of intermediate home stops between any two activities, and then have the algorithm drop intermediate home stops and/or low priority activities. The priorities could even be coupled to household or weekly agendas, discussed below.

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

- The present model for the estimation of travel times between different locations is not very realistic. It would be possible to use travel times from a simulation in order to increase realism. With time dependent travel times, phenomena like congestion avoidance within the day plans should emerge.
- The paper pretends that all activity scheduling problems can be solved with a single utility function. This is improbable, and variations of the relevant coefficients could be easily introduced. Such coefficients could for example be obtained from estimated econometric models of daily activity schedules (e.g. Jara-Diaz et al. 2003). That paper also explains how such econometric explanations relate to random utility theory.
- Aspects of limited information could be modeled by only making subsets of possible locations available to the algorithm. Such sets would need to be generated by other methods, for example by random sampling, by information transmission via sodkistler (Marchal & Nagel 2005), or by using mental maps (Arentze & Timmermans 2003; Kistler 2004).
- The paper assumes that each agent optimizes for him-/herself. This reflects our observation that simple copying of partial strategies from other people does not work in the real world. For example, I cannot simply copy the shopping location from my neighbor because in all probability he/she works somewhere else and so this may not be convenient. A possible approach might once more be the use of social networks (Marchal & Nagel 2005) which ensures that information is primarily passed between people who share some similarities.
- One interesting aspect of activity planning is to generate activity plans for households and/or for full weeks instead of generating them for single agents and single days. This problem is more complex than simply generating individual day plans for each of the occupants of a household since the activity plans of different occupants or of consecutive days depend on each other. For instance, only one of the agents living in a household has to go shopping, or shopping only needs to be done once a week. An investigation of this has been done by Meister et al. (2004).

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

9. Summary

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

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

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

Acknowledgement

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

Notes

1. In this paper, the word 'parents' is only used in the computational sense, never in the biological sense.
2. The second if-condition is only relevant if β_{wait} is different from zero. As will be discussed in Section 8, it makes much sense to set it to zero, in which case this condition is no longer relevant.

References

Abraham J & Hunt J (2002) *Spatial Market Representations: Concepts and Application to Integrated Planning Models*. Presented at the 49th annual North American Meetings of the Regional Science Association International, San Juan, Puerto Rico.

Arentze T, Hofman F, van Mourik H & Timmermans H (2000) *ALBATROSS: A Multi-agent Rule-based Model of Activity Pattern Decisions*. Paper 00-0022, Transportation Research Board Annual Meeting, Washington, D.C.

Arentze T & Timmermans H (2003) Representing mental maps and cognitive learning in micro-simulation models of activity-travel choice dynamics'. In: *Proceedings of the*

meeting of the International Association for Travel Behavior Research (IATBR). Lucerne, Switzerland. See www.ivt.baum.ethz.ch.

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

Beckman RJ, Baggerly KA & McKay MD (1996) Creating synthetic base-line populations. *Transportion Research Part A – Policy and Practice* 30(6), 415–429.

Ben-Akiva M & Lerman SR (1985) *Discrete Choice Analysis*. Cambridge, MA: The MIT Press.

Bhat C, Guo J, Srinivasan S & Sivakumar A forthcoming. A comprehensive econometric microsimulator for daily activity-travel patterns. *Transportation Research Record*. Also see www.ce.utexas.edu/prof/bhat/FULL_REPORTS.htm.

Bowman JL (1998) The Day Activity Schedule Approach to Travel Demand Analysis. Ph.D. thesis Cambridge, MA: Massachusetts Institute of Technology.

Bowman J, Bradley M, Shiftan Y, Lawton T & Ben-Akiva M (1999) Demonstration of an activity-based model for Portland. In: *World Transport Research: Selected Proceedings of the 8th World Conference on Transport Research 1998*, Vol. 3. (pp. 171–184.) Oxford: Elsevier.

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

Doherty ST & Axhausen KW (1998) The development of a unified modelling framework for the household activity-travel scheduling process. In: *Verkehr und Mobilität*, Vol. 66 of *''Stadt Region Land'', OPTchapter = , OPTtype = , address = Technical University, Aachen, Germany, OPTedition = , OPTmonth = , OPTpages = , OPTnote = , OPTannote =* . Institut für Stadtbauwesen.

DYNASMART (www page: accessed 2003). See www.dynasmart.com and dynamictraffic-assignment.org.

EIRASS'04 (2004) *Workshop on Progress in Activity-based Analysis*. Maastricht, NL.

IATBR'03 (2003) *Proceedings of the Meeting of the International Association for Travel Behavior Research (IATBR)'*. Lucerne, Switzerland: See http://www.ivt.baum.ethz.ch/allgemein/iatbr2003.html.

Jara-Diaz et al. (2003) Modelling activity duration and travel choice from a common microeconomic framework. In: *Proceedings of the Meeting of the International Association for Travel Behavior Research (IATBR)*. Lucerne, Switzerland. See www.ivt.baum.ethz.ch.

Kaufsfman DE, Wunderlich KE & Smith RL (1991) *An Iterative Routing/Assignment Method for Anticipatory Real-Time Route Guidance*. Technical Report IVHS Technical Report 91–02, University of Michigan, Department of Industrial and Operations Engineering, Ann Arbor MI 48109.

Kistler D (2004) *Mental Maps for Mobility Simulations of Agents*. Master's thesis, ETH Zurich.

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*. See http://tmip.fhwa.dot.gov/clearinghouse/docs/abtf/.

Marchal F & Nagel K (2005) *Modelling Location Choice of secondary Activities with a Social Network of Cooperative Agents*. Paper, Transportation Research Board Annual Meeting, Washington, D.C.

MATSIM (www page: accessed 2004), *MultiAgent Transportation SIMulation*. See www.matsim.org.

Meister K, Frick M & Axhausen K (2004) *A GA-based Household Scheduler*. Arbeitsbericht Verkehrs- und Raumplanung 241, Institute for Transport Planning and Systems, ETH Zurich, Switzerland. See www.ivt.baug.ethz.ch. Submitted to TRB'05.

Miller E & Roorda M (2003) A prototype model of household activity/travel scheduling. *Transportation Research Record* 1831: 114–121.

Nagel K & Barrett C (1997) Using microsimulation feedback for trip adaptation for realistic traffic in Dallas. *International Journal of Modern Physics C* 8(3): 505–526.

Pendyala R accessed (2004) *Phased Implementation of a Multimodal Activity-Based Travel Demand Modeling System in Florida*. See www.dot.state.fl.us/research-center/Completed_PTO.htm. Project number 0510812 (BA496).

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

TRANSIMS (www page: accessed 2004) *TRansportation ANalysis and SIMulation System*. transims.tsasa.lanl.gov. Los Alamos National Laboratory, Los Alamos, NM.

About the authors

David Charypar was born on 3 February 1978 in Urdorf, Switzerland. He received his Dipl. degree in computer science from the Swiss Federal Institute of Technology (ETH), Switzerland in 2003. 2003–2004, he was a Ph.D. student in the group for Modeling and Simulation of Prof. Kai Nagel at ETH Zurich. Since 2004, he is a Ph.D. student at the Computational Science and Engineering Laboratory of Prof. Petros Koumoutsakos, ETH Zurich.

Kai Nagel was born 17 September 1965 in Cologne, Germany. Education in physics, meteorology, and computer science in Cologne (Germany) and Paris. 1994 Ph.D. in computer science. 1995–1999 Los Alamos National Laboratory. 1999–2004 assistant professor for Computer Science at ETH Zurich. Since 2004 full professor for "Transport systems planning and transport telematics" at the Technical University of Berlin.