Q-learning for flexible learning of daily activity plans

David Charypar, Dept. of Computer Science, ETH Zürich
Philip Graf, Dept. of Computer Science, ETH Zürich
Kai Nagel, Dept. of Computer Science, ETH Zürich

Conference paper STRC 2004

STRC

4th Swiss Transport Research Conference
Monte Verità / Ascona, March 25-26, 2004

Q-learning for flexible learning of daily activity plans

David Charypar, Philip Graf
Dept. of Computer Science
ETH Zürich
Zürich, Switzerland

Phone: 01-632 47 63
Fax: 01-632 13 74
eMail: charypar@inf.ethz.ch

Kai Nagel
Dept. of Computer Science
ETH Zürich
Zürichr, Switzerland

Phone: 01-632 54 27
Fax: 01-632 13 74
eMail: nagel@inf.ethz.ch

Abstract

Q-learning is a method from artificial intelligence to solve the reinforcement learning problem (RLP). The RLP is defined as follows: An agent is faced with a set of states, S. For each state there is a set of actions, A_s, which the agent can take. At each state, the agent receives a (possibly negative) reward. The task of the agent is to select its actions such that it maximizes its reward.

Activity generation is a technique used for demand generation in the context of transportation simulation. For each member of a synthetic population, a daily activity plan needs to be found, stating for each individual a sequence of activities (e.g. home-work-shop-leisure-home), their locations, and their times. Activities at different locations then generate demand for transportation.

Activity generation can be modeled as a reinforcement learning problem by assuming that the states are given by the triple (type of activity, starting time of activity and time already spent at activity). The possible actions are either to stay at a given activity, or to move to another activity. Rewards are given as "utility per time slice", which corresponds to a coarse version of marginal utility. The task of the agent is to find "circles" in this 24-hour-periodic state space that maximize the daily utility. Q-learning has the property that, by repeating similar experiences over and over again, the agent also looks forward in time, i.e. the agent can also go on paths through state space where high rewards are only given at the end. Another advantage of Q-learning is that it also explores the state space away from the optimal path. In consequence, an agent that is thrown off the optimal path is still able to make good decisions based on the Q-values.

This paper presents computational results with such an algorithm for daily activity planning.

Keywords

Activity Planning – Q-Learning – Activity Generation – Within Day Re-learning – Multi Agent Travel Simulation – 4th Swiss Transport Research Conference – STRC 04 – Monte Verità

1. Introduction

It is a recent trend in transportation research to use the problem of activity planning in order to generate demand for transportation. Transportation demand is naturally derived from performing activities at different locations. The larger context of this work is the push towards 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. A synthetic population is a Monte-Carlo realization of the information provided by the census data.
- Next, synthetic **activity plans** are generated for each member of the population. This can be done in several different ways with several different levels of complexity and flexibility. Flexible all-day activity plans are the topic of this paper.
- Once the activity plans are known, each individual plans the travel that connects activities at different locations, including **mode choice** and **route choice**.
- 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 **traffic (micro-)simulation**, all agents' plans are simultaneously executed and the results of interaction are computed. One important interaction result is congestion.
- As is well known, the causal relation between the modules goes into both directions. For example, if many agents chose 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. This can be either interpreted as relaxation or as human learning.

If the above process is successful, after many iterations, we end up with each agent in the simulation having one (or several) reasonable plan(s) that the agent executes. These plans altogether approximate an equilibrium of the simulation. In this equilibrium one expects that the sum of all travel times is plausibly low and that the agents carry out activities in a useful, reasonable way.

For this approach, many collaborating modules need to be designed, implemented, and tested. Here, we concentrate on a module to generate activity plans. More specifically, the module will generate the **time allocation** part of activity plans, i.e. when activities should begin, how long they should last, and when they should end. The method should, however, also be capable to do location choice, or activity pattern selection.

2. Q-Learning

In artificial intelligence, Q-learning is a well know algorithm that solves the reinforcement learning problem. The reinforcement learning problem (RLP) can be stated as follows: Given a set S of states s, transitions between states $s \rightarrow s'$, and rewards $R(s \rightarrow s')$ associated with each transition. At each state, the agent can select between different actions $a \in A(s)$, which influence the transition probabilities between states. The task of the agent is to select actions $a(s)$ such that some averaged discounted expected reward, $\sum_t \beta^t R_t$, is maximized. $\beta < 1$ is the discount factor, and R_t is the reward being obtained at each time step t.

Rewards can be high, low or even negative. They may come with a delay, in the sense that some transition may not lead to immediate high reward, but possibly to a high later reward which cannot be reached by any other sequence of transitions. The reward for a particular action in a given state may be very high but the series of actions that leads to that state may have a very low or even negative reward.

β models the effect of how far the agent looks into the future. A β close to one means that rewards in the far future carry large weight; a small β means the opposite. Somewhat more precisely, β models the trade-off between getting some reward now and a higher reward later. It makes sense for human behavior to assume that a low reward at some point in time is only accepted if this is over-compensated at some later time.

Q-learning (e.g. Russel and Norvig, 1995, pp. 598–624) is a method to solve the RLP. In Q-learning an agent learns action-values giving the expected utility of taking a given action in a given state. These action-values are also called Q-values. In each state the agent has several options for actions it could execute. For each state action pair, the agent stores an individual Q-value that is used for the decision process. The Q-value for a given state action pair corresponds to the expected cumulative reward that can be collected by taking the action. This expected cumulative reward depends on all rewards that will be collected from that moment on.

Q-values are defined the following way:

$$Q_\infty(s,a) = R(s,a) + \beta \max_{a'} Q_\infty(s',a'), \tag{1}$$

where $R(s,a)$ is the reward for taking action a in state s and s' is the state that is the result of executing action a in state s. If the transition after taking a is probabilistic (i.e. several different s' can result), then some suitable average needs to be taken. β is again the discount parameter, in the range $[0..1[$. In words that is the $Q_\infty(s,a)$ equals the reward given for the state action pair (s,a) plus β times the maximal expected cumulative reward in the resulting state s'.

If an agent always takes the action $a' \in A(s')$ which maximizes $Q(s',a')$, then $Q_\infty(s,a)$ is exactly the (expected) reward $\sum_t \beta^t R_t$ when starting in state s. When deviating from that "path", the reward will be reduced. This shows that always taking the action with the highest Q value solves the RLP.

Note that for low discount parameters, Q_∞ depends almost entirely on the current state action

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

$$\tilde{Q}_\infty = \frac{1}{1-\beta}\,\bar{R}\,. \tag{2}$$

By this argument, one sees that the final Q-values are proportional to the average reward, and the proportionality factor is $1/(1-\beta)$. Note that for $\beta \lesssim 1$, the factor is very large, and for $\beta \to 1$, it goes to infinity.

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

1. Initialize the Q-values.
2. Select a random starting state s which has at least one possible action to select from.
3. Select one of the possible actions. This action will get you to the next state s'.
4. Update the Q-value of the state action pair (s, a) according to the update rule (3) below.
5. Let $s = s'$ and continue with step 3 if the new state has a least one possible action. If it has none go to step 2.

The update rule is given by

$$Q_{t+1}(s,a) = (1-\alpha)\,Q_t(s,a) + \alpha\,[R(s,a) + \beta \max_{a'} Q_t(s',a')], \tag{3}$$

where $Q_t(s,a)$ is the Q-value at the current time-step and $Q_{t+1}(s,a)$ is the updated value. α is the learning rate and is a parameter of the algorithm.

The proper selection of α is crucial in many applications. Watkins and Dayan (1992) showed that the Q-learning algorithm converges to the steady state if the following preconditions are met:

- The learning rate α lies within $]0..1]$.
- The learning rate decreases with each evaluation of the update rule, i.e. $\alpha_1 > \alpha_2 > \alpha_3 \ldots$

- The following conditions are met:

$$\sum_{i=1}^{\infty} \alpha_i = \infty \tag{4}$$

and

$$\sum_{i=1}^{\infty} \alpha_i^2 < \infty. \tag{5}$$

One way of achieving this is by setting $\alpha_i = c/i$, where c is a constant.

However, these conditions are only important if the reward function is noisy. For completely deterministic rewards one can use $\alpha_i = 1$ in order to learn as fast as possible[1].

Q-learning is an active learning algorithm. As a consequence, the learning structure (i.e. which state action pairs are visited) depends solely on the algorithm's choice. If the algorithm selects the right actions, we can expect a good learning performance. On the other hand, bad action selection policies can lead to very poor results.

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

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

Now assume that the agent has learned some Q values for this situation, either Q_∞ according to Eq. (1) or some other values. Assume that now exploration is switched off (i.e. $p_{explore} = 0$), that is, at every state s the agent selects the action a which maximizes $Q(s, a)$. Can one say anything about the long-term behavior in this situation?

In fact, this describes a discrete dynamical system. Let us also assume that the number of state action pairs is finite. For such a system, one can say the following:

- If the system is **deterministic**, then the trajectory needs to go to an attractor, which is either a fixed point or cyclic.[2]

[1] There may be some fluctuations at the beginning of the learning process. However, we never observed anything like that.

[2] Since the system is discrete and finite, the trajectory eventually needs to come back to a state where it was before. Since the system is deterministic, from then on it will do exactly the same as in the previous "round".

- If the system is **stochastic**, then the system (since it is Markovian) will go to a steady state phase space density. There may, however, be a flow in phase space, resembling a stochastic version of the cyclic attractor of the deterministic system.

3. Q learning for daily activities of humans

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

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

The time structure is assumed to be periodic, that is, at $t = T$ the agent is connected to $t = 1$, and over the transition it can stay or switch as it can do with any time increment.

Now assume once more that the agent has learned some Q values and now does exploitation only, i.e. it always choses the action a which maximizes $Q(s, a)$ at any state s. Because of the time structure, the system cannot go to a fixed point, and so under normal circumstances it will describe a cycle through 24-hour state space. If the RLP was completely solved by the agent, then that path maximizes the score (utility) per 24 hours.[3] If the RLP was not completely solved, i.e. some of the Q values do not correspond to the steady state values, then the agent will nevertheless find a cycle, albeit possibly not the optimal one.

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

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

[3] That statement is correct only for $\beta \to 1$. For smaller β, the situation is similar, but not exactly the same.

Since we are interested in large scale scenarios with up to 10 million agents, the computation of the plans should not need more than a small number of seconds per agent. Even with such relatively short computing times per agent, multi-agent simulations for scenarios of that size use parallel computing for computational speed. It is difficult to implement within-day replanning in such a parallel system without giving up parallel performance and module pluggability (Nagel and Marchal, 2003). For that reason, most if not all large scale multi-agent transportation simulations at this point need the complete all-day activity plan *pre*-planned before the simulated day starts. However, even though a large part of the traffic we encounter in our daily life can be described in this static way, there is a large and important part that cannot be described like this. There are unforeseen things that happen: Accidents lead to increased travel times on some links, a shop may be closed at arrival if one has too much unexpected congestion on the way there, or cars may break down. In other words, just following a simple pre-computed plan is not realistic.

Taking these last two arguments together, one notices that Q-learning in fact solves the within-day replanning problem without having to explicitly re-plan. Instead of the deterministic plan, a table with the relevant Q-values would be passed on to the traffic (micro-)simulation. The traffic micro-simulation would then decide for each agent when to leave activities etc. based on those Q-values.

4. Synthetic Example

We find it important to test a method using synthetic examples when we apply it to a problem for the first time. On one hand, this gives first results that are much more significant in pointing at issues in conjunction with the new method, on the other hand, it gives some certainty that the implementation will do what we want.

For testing purposes we designed the following task: Given an activity pattern of 4 activities, “Home”, “Work”, “Shop” and “Leisure” we want to solve the time allocation problem. E.g. find starting and ending times for these 4 activities such that the resulting overall utility is maximal. The overall utility is thereby defined as the sum of the utilities of the individual activities.

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

Throughout all our preliminary studies we use a utility per hour of 1. This results in longer activities having higher maximal utilities. If we would set the maximal utility of all activities to the same value, e.g. 10, the optimal solution to the time allocation problem would be to skip (i.e. assign as little time as possible to) the long activities and only perform the short ones.

The activities “Shop” and “Leisure” both use the same utility function. It is of the same type as the one of “Home” but the step is at 2 instead of 7 hours. Also the height of the step is set to 2 instead of 7 complying our “1 util per hour”-rule.

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

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

In a certain sense, the described utility functions represent the worst case for a utility landscape. However, this utility functions *do not* model reality. Real utility functions generally are much smoother and give rewards already at earlier times of execution.

In order to make the Q-learning approach feasible we have to discretize the time. This we do using time slots of 60, 30 or 15 minutes respectively.

We are focusing on the time allocation and want to leave the order of the activities the same. Therefore, exchanging activities is not an option. For this reason, in every time slot the agent can only decide whether it would like to stay at the current activity or change to the next.

In order to keep the number of states finite, we restrict the execution time of an activity to 12 hours. Depending on the resolution this is equivalent to setting the maximal number of consecutive states spend in the same activity to 12, 24 or 48 respectively.

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

To account for the fact that an agent usually has to travel from one location to an other between activities we define a constant travel time between different activities. We choose it to be 60 minutes. The utility of travel is set to be zero. One might argue that the utility of travel should be negative. However, as all our activities have positive utilities per time it is already desirable to minimize travel times.

As mentioned earlier, there are basically 3 parameters playing a role for Q-learning: The discount parameter β, the learning rate α and the exploration rate $p_{explore}$.

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

Q-values.

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

Knowing all that, one has to find a compromise. Our preliminary tests showed that for time-slots of 60 minutes a discount parameter of $\beta = 0.96$ works fine. Note that as the resolution increases the length of the activities in terms of number of states also increases and, as a consequence, the discount parameter β has to be increased accordingly.

We want the discount per time t to be the same for all times $t > 0$ independent of the time resolution t_{res}.

$$\beta_{res_1}^{(\frac{t}{t_{res_1}})} = \beta_{res_2}^{(\frac{t}{t_{res_2}})} \tag{6}$$

This leads to

$$\beta_{res_2} = \beta_{res_1}^{(\frac{t_{res_2}}{t_{res_1}})}. \tag{7}$$

For β_{res_1} close to 1.0 and $t_{res_2} < t_{res_1}$, we can approximate β_{res_2} by

$$\beta_{res_2} \approx 1 - (1 - \beta_{res_1}) \frac{t_{res_2}}{t_{res_1}}. \tag{8}$$

By setting $\beta_{res_1} = 0.96$ and $t_{res_1} = 60\text{min}$ we get

$$\beta_{res_2} = 1 - \frac{0.04\, t_{res_2}}{60\text{min}}, \tag{9}$$

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

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

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

"High" and "low" are defined with respect to the final Q values $Q_\infty(s, a)$: A high initial Q value is larger than any final Q_∞, a low initial Q value is smaller than any final Q_∞. Some a priori estimates can be obtained from Eq. (2): Q values are certainly high when they are above $1/(1-\beta)\, R_{max}$, where R_{max} is the largest reward in the system. Q values are certainly low when they are below $1/(1-\beta)\, R_{min}$, where R_{min} is the smallest reward in the system. This also makes clear that "high" and "low" depend on the particular value of β that was selected.

Table 1: The optimal day's plan for the synthetic example with 9 hours "Work" that only pays of if started at 8:00, 2 hours floating "Shop" and "Leisure" and 7 hours floating "Home". Travel times between activities are set to 60 minutes.

Start time - End time	Activity
08:00 - 17:00	Work
18:00 - 20:00	Shop
21:00 - 23:00	Leisure
00:00 - 07:00	Home

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

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

5. Results for synthetic example

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

In order to keep the discount per time nearly constant we applied equation (9) for all time resolutions resulting in β values of 0.96, 0.98 and 0.99 respectively.

From the design of the utility functions for the four activities "Work", "Shop", "Leisure" and "Home", it follows that the optimal daily plan corresponds to the one shown in table 1. Note that the time between activities is needed for traveling from one location to another.

We now compare the solutions found by the algorithm with the optimal solution shown in table 1. On top of that, we look at the learned Q-values and appraise the ability of the solution to recover from disturbances. Only if fast and plausible ways of recovering are observed we assume that the algorithm has converged to a good solution.

In table 2 we show an overview of the results obtained by our tests. It can be seen that doubling the resolution leads to an increase in the number of iterations needed to converge by a factor of 10. The table gives the minimal number of iterations needed in order to converge with a probability higher than 50%. We considered the algorithm as having converged if both the optimal daily plan

Table 2: Results of tests with the synthetic example using high initialization. We indicate the number of iterations needed to converge to the optimal solution with a probability substantially higher than 50%. See table 1 for the optimal solution.

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

corresponded to the one shown in table 1 and the agent was able to recover from disturbances in a reasonable matter.

The running times were measured on a Mobile Pentium 4 with 2.4 GHz. Our programs were implemented using Java.

One might worry about reliability of the algorithm. As was already mentioned, table 2 says only something about then number of iterations needed in order to converge with a probability of *at least* 50%. What if we need a system that converges in at least 99% of the cases? It might be necessaray to increase the number of iterations to 10 times the indicated value or even more. Fortunately, it does not seem to be this way. In fact, we never observed a failure to converge if the algorithm was run for twice the number of iterations stated in the table.

To us, it seems to be important to point out that result of the Q-learning algorithm is not only an optimal daily plan. The result, in fact, are optimal daily plans when started at an arbitrary state. This makes it possible for an agent in a dynamic travel simulation to react to unforeseen events in a very short time. Actually, almost no time is needed to calculate the reaction to a disturbance as the agent only has to look at the Q-values and identify the action with the maximum Q-value for each state that the agent visits.

6. More realistic reward tables

The data used for our synthetic tests are not very realistic.

The synthetic test were explicitly designed to be difficult to solve. In a certain sense they represent the worst case of a distribution of rewards as a reward for a particular activity is always given only at the end of the activity. So, in order to find a good daily plan the agent has to look very far in the future.

Looking far into the future is equivalent to discount values β close to one, and large discount value make it harder to converge. Therefore out synthetic test are well suited to test the algorithm.

Table 3: Travel times for the real world example. We assume constant travel times during the whole day.

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

However, real life is different. Real activities already give rewards at earlier times. For example, being at home pays off already very early which corresponds to a reward table for the activity "Home" that has some positive rewards for each hour that it is executed. As a result of the above reflections we introduce new reward tables for all of the four activities. These should correspond better to what we observe in real life than the synthetic reward tables from the first tests. We now point out the characteristics of the 4 reward tables:

- Activity "Home": The reward for spending one hour at home is independent of the time that was already spent there. The reward only depends on the time of day, assuming that being at home during the night (sleeping) pays off more than being at during the day. See figure 1.
- Activity "Work": Work pays off most from 8:00 until 18:00. If the agent performs work outside this time window the rewards per time slice get gradually reduced. The agent gets a bonus for staying at work for more than 9 hours. However, the reward is reduced if 10 or more hours are spent at work. Working between 21:00 and 7:00 does not give any reward at all. See figure 2.
- Activity "Shop": We assume that shops are open from 8:00 until 19:00. Therefore, shopping gives only rewards during the day. As we are looking at daily plans, we set the maximal shopping time to 45 minutes. If an agent shops for a longer time, it does not get any reward for the additional time. See figure 3.
- Activity "Leisure": We define leisure similar to activity "Home". The reward for having one time slice of leisure is maximal from 19:00 until 24:00 and independent of the time already spent in this activity. It is minimal, although not zero, from 5:30 until 13:00. There is a smooth transition between these two extrema. See figure 4.

On top of changing the rewards for the different activities, we changed the travel times between the different activities. Please find the new travel times in table 3.

As with the synthetic example, we test the algorithm with the new reward tables and travel times with a time resolution of 60, 30 and 15 minutes. All Q-learning parameters were set to the same

Figure 1: Plot of the rewards per 15 minutes for activity “Home” in the real world example. Rewards depend on the starting time of an activity and the duration that it lasted so far. Starting time is running from left to right (0h–24h), duration from bottom to top (0h–12h). Dark blue corresponds to no reward (0), dark red corresponds to maximal reward (10).

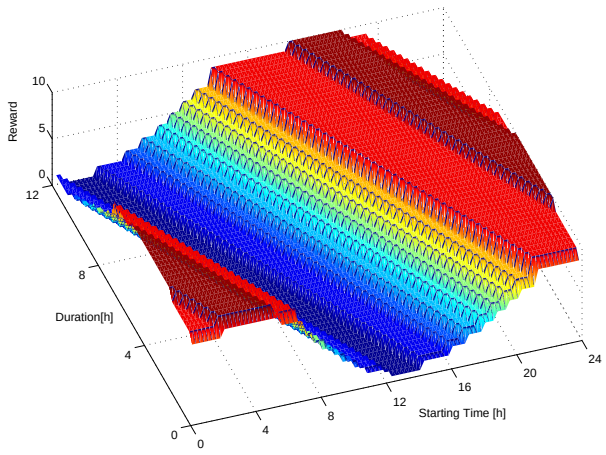

Figure 2: Plot of the rewards per 15 minutes for activity “Work” in the real world example. Rewards depend on the starting time of an activity and the duration that it lasted so far. Starting time is running from left to right (0h–24h), duration from bottom to top (0h–12h). Dark blue corresponds to no reward (0), dark red corresponds to maximal reward. In this plot the maximal reward is *30* which is higher than in the other plots.

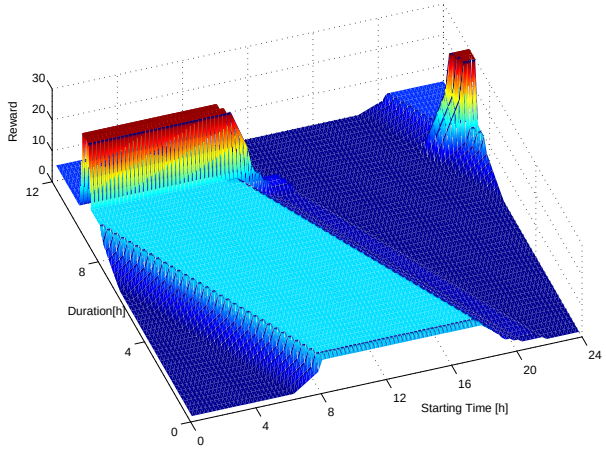

Figure 3: Plot of the rewards per 15 minutes for activity “Shop” in the real world example. Rewards depend on the starting time of an activity and the duration that it lasted so far. Starting time is running from left to right (0h–24h), duration from bottom to top (0h–12h). Dark blue corresponds to no reward (0), dark red corresponds to maximal reward (10).

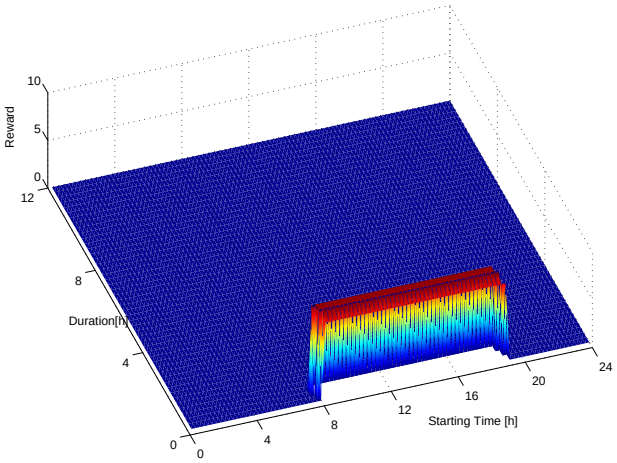

Figure 4: Plot of the rewards per 15 minutes for activity “Leisure” in the real world example. Rewards depend on the starting time of an activity and the duration that it lasted so far. Starting time is running from left to right (0h–24h), duration from bottom to top (0h–12h). Dark blue corresponds to no reward (0), dark red corresponds to maximal reward (10).

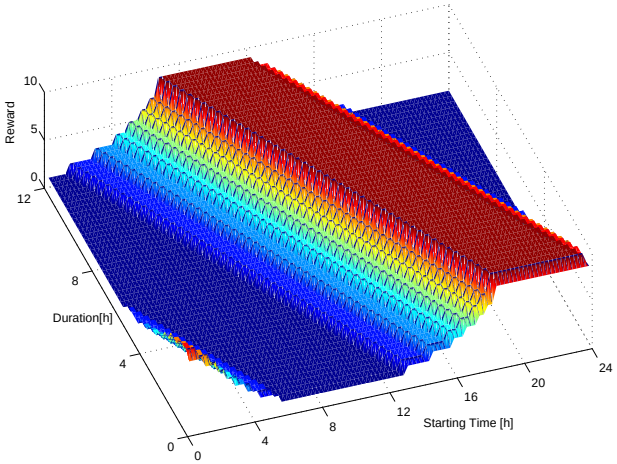

Table 4: Best daily plan for time resolution 60 minutes. The time between the end of one activity and the beginning of the next activity is spent traveling.

Start time - End time	Activity
08:00 - 17:00	Work
18:00 - 18:00	Shop (skipped)
19:00 - 23:00	Leisure
00:00 - 07:00	Home

values as in the synthetic example. As the long term goal, we would like to generate daily activity plans for about 10 million agents in a large simulation. As a consequence, available processor time for each agent is very limited. As a rough estimate, the generation of a daily plan for one agent should not take longer than one second. In search of a planning algorithm which is as fast as possible we also tried to use a low initialization approach. As mentioned earlier, Q-learning with low initialization quickly finds feasible solutions at the expense of slower convergence to the optimal solution. So if one is looking for reasonable good results and does not depend on optimal solutions this might be the way to go.

With low initialization, we have to make sure that exploration is taken care of by other means. We therefore use higher exploration rates in this case as is also indicated in table 8. Depending on the time resolution we use a exploration rate $p_{explore}$ of 0.4, 0.2 or 0.1 respectively.

7. Results for Real World

First we ran the Q-learning algorithm multiple times for a long time in order to reliably find a good daily plan for the given reward tables. This was done for each resolution independently. The resulting solutions are shown in tables 4, 5 and 6 respectively. Further on, we will refer to these solutions as the optimal solutions.

Similar to the synthetic tests, we test the algorithm for time resolutions of 60, 30 and 15 minutes respectively and identify the number of iterations needed in order to converge to the best solution with a probability greater than 50%. This we do both for the high initialization approach and the low initialization approach. See table 7 for results using the high initialization and table 8 for results of the low initialization approach.

The number of iterations necessary to converge for the real world example with high initialization and the synthetic examples are almost the same. This seems to make sense as the high initialization results in a complete exploration of the state space. As the state space is of the same size in both cases, it is to be expected that the number of iterations needed to explore it is roughly

Table 5: Best daily plan for time resolution 30 minutes. The time between the end of one activity and the beginning of the next activity is spent traveling.

Start time - End time	Activity
08:00 - 17:30	Work
18:00 - 18:30	Shop
19:00 - 23:30	Leisure
00:00 - 07:00	Home

Table 6: Best daily plan for time resolution 15 minutes. The time between the end of one activity and the beginning of the next activity is spent traveling.

Start time - End time	Activity
08:00 - 17:30	Work
17:45 - 18:15	Shop
18:30 - 23:45	Leisure
00:15 - 07:15	Home

Table 7: Results of real world example with high initialization. We indicate the number of iterations needed to converge to the optimal solution with a probability substantially higher than 50%.

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

Table 8: Results of real world tests with low initialization. We indicate the number of iterations needed to converge to a reasonable solution with a probability substantially higher than 50%.

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

the same. Only at the highest time resolution there is an observable difference: The real world example converges to the best solution already after 2 million iterations compared to 5 million iterations for the synthetic example.

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

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

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

8. Discussion and further work

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

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

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

If we assume that a fictitious algorithm would know all the optimal paths in the state space in advance (i.e. it would know in each state which action maximizes the cumulative discounted reward) it would need 229 state transitions to calculate the proper Q-value for that state if it was restricted not to look at other Q-values that it might have calculated before. This algorithm would need $n_{Q-values} \cdot t_{\mathrm{horizon}_{10\%}}$ steps to identify all the Q-values. For our case this equals to a cumulative number of $36864 \cdot 229 = 8.4\mathrm{mill}$ steps.

However, a real algorithm probably will take advantage of the interdependence of the Q-values and it might take advantage of the structure of the reward tables or the fact that we are only looking for optimal daily plans — and not for the exact Q-values — to solve the problem more quickly. Please note, that our Q-learning algorithm needs only $2\mathrm{mill}$ visits to come up with the optimal flexible daily plan. We assume that this is because of the fact that our initialization is quite close to the final Q-values and that we were not asking for the Q-values to have the final value but only that they lead to the right decisions. Also, our algorithm does not guarantee that the solution found is the optimal one.

We believe that the problem of finding an optimal cyclic 24 hours plan together with all reactions to disturbances can be solved using some sort of adapted shortest path algorithm (e.g. Dijkstra's). Our future studies will investigate that idea.

One problem with our implementation of Q-learning is the need for reward tables. These have a large number of values to be defined (for $t_{res} = 15\mathrm{min}$ there are 18432). The question is where to get these from. We suggest to use the inherent structure in realistic reward tables to fill them in using some sort of function. An other option is to use this function directly to give the rewards. This eliminates the need for tables and therefore reduces memory consumption. However, depending on the kind of reward function one wants to use, this approach might have lower computational efficiency.

We intend to use reward tables that are initialized according to the utility functions proposed in (Charypar and Nagel, 2003). We hope that this will lead to similarly good results as the ones presented here, while eliminating the need to explicitly set many independent values in the reward

tables. That utility function has only very few parameters that have to be specified allowing it to get usable reward tables very quickly.

It seems that finding a good time allocation for a given activity pattern is well feasible for our Q-learning algorithm. Because of that, we want to try also the case where the algorithm has the freedom to choose which activity should be next. This would give even more flexibility as it would also be an option to drop an activity if the agent late.

9. Summary

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

The solution found is not only an optimal daily plan but it also holds all information necessary to react to unforeseen disturbances. If such a disturbance occurs, all the agent which uses the daily plan has to do in order to react is to look up in a table the best actions to take in that case.

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

Using this method in a multi agent travel simulation to plan the activities of up to 10 mill agents seems feasible and the resulting within day re-planning capabilities promise new approaches in realism.

References

Charypar, D. and K. Nagel (2003) Generating complete all-day activity plans with genetic algorithms, in *Proceedings of the meeting of the International Association for Travel Behavior Research (IATBR)*, Lucerne, Switzerland, August 2003. See http://www.ivt.baum.ethz.ch/allgemein/iatbr2003.html.

Nagel, K. and F. Marchal (2003) Computational methods for multi-agent simulations of travel behavior, in *Proceedings of the meeting of the International Association for Travel Behavior Research (IATBR)*, Lucerne, Switzerland, August 2003. See http://www.ivt.baum.ethz.ch/allgemein/iatbr2003.html.

Russel, S. and P. Norvig (1995) *Artificial intelligence: a modern approach*, Series in Artificial Intelligence, Prentice Hall.

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