Q-learning for flexible learning of daily activity plans

David Charypar
Dept. of Computer Science, ETH Zürich, Switzerland
E-mail: charypar@inf.ethz.ch

Kai Nagel (corresponding author)
Inst. for Land and Sea Transport Systems, TU Berlin, Germany
Tel: +49-30-314-23308
Fax: +49-30-314-26269
E-mail: nagel@vsp.tu-berlin.de, nagel@kainagel.org

Submission date: 1 April 2005

Number of words: 6559 + 2 Tables + 1 Figure

ABSTRACT

Q-learning is a method from artificial intelligence to solve the reinforcement learning problem (RLP), defined as follows: An agent is faced with a set of states, S. For each state s there is a set of actions, $A(s)$, which the agent can take, and which takes the agent (deterministically or stochastically) to another state. For each state, the agent receives a (possibly stochastic) reward. The task is to select actions such that the reward is maximized.

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

Activity generation can be modeled as RLP with the states given by the triple (type of activity, starting time of activity, time already spent at activity). The possible actions are either to stay at a given activity, or to move to another activity. Rewards are given as "utility per time slice", which corresponds to a coarse version of marginal utility. Q-learning has the property that, by repeating similar experiences over and over again, the agent looks forward in time, i.e. the agent can also go on paths through state space where high rewards are only given at the end. This paper presents computational results with such an algorithm for daily activity planning.

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

1 INTRODUCTION

It is a recent trend in transportation research to use activities in order to generate demand for transportation. Transportation demand is naturally derived from performing activities at different locations. For each synthetic individual, a sequence of activities is generated, including activity location and activity times. Activity-based demand generation is a very active field of research; see, e.g., the proceedings of the two recent conferences (*1, 2*). The mainstay of activity-based demand generation are random utility models (RUMs) (*3, 4, 5, 6*). RUMs, however, arguably have the disadvantage that (a) they are behaviorally not very realistic, (b) only heuristic approaches exist for reducing the "choice set" when faced with a very large number of options, (c) they need to be re-computed when a traveler is thrown "off" its optimal dayplan. In consequence, alternatives are also investigated, such as behaviorally or rule-based approaches (e.g. (*7, 8*)).

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

This paper concentrates on a module to generate the *time allocation* part of activity plans, i.e. when activities should begin, how long they should last, and when they should end. The method should, however, also be capable to do location choice, or activity pattern selection. Some discussion of this, and related methods that could be explored, can be found in Section 6.

This paper starts with a review of Q-learning (Section 2), followed by a section on how to apply this to activity generation (Section 3). The method is first tested on an unrealistic but somewhat challenging "test" example, and then with a somewhat more realistic case (Section 5). The paper is concluded by a discussion (Section 6) and a summary (Section 7).

2 Q-LEARNING

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

$$E\left(\sum_t \beta^t R_t\right), \tag{1}$$

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

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

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

Q-learning (e.g. (*9*)) is a method to solve the RLP. In Q-learning an agent learns action-values giving the expected utility of taking a certain action in a given state. These

action-values are also called Q-values. In each state the agent has several options for actions it could execute. For each state-action-pair (s,a), the agent stores an individual Q-value $Q(s,a)$ that is used for the decision process. The Q-value for a given state-action-pair corresponds to the expected cumulative reward, Eq. (1), that can be collected by taking the action.

Q-values are defined the following way:

$$Q_\infty(s,a) = R(s') + \beta \max_{a'} Q_\infty(s',a'), \tag{2}$$

where $R(s')$ is the reward for arriving at s', and s' is the state that is the result of executing action a in state s. If the transition after taking 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[.

Eq. (2), when expanded, leads to a solution of type $\sum_t \beta^t R_t$. If an agent always takes the action $a \in A(s)$ which maximizes $Q_\infty(s,a)$, then the agent maximizes that sum. When deviating from that "path", the reward will be reduced. This shows that always taking the action with the highest Q-value solves the RLP.

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

$$\widetilde{Q}_\infty = \frac{1}{1-\beta}\overline{R}. \tag{3}$$

By this argument, one sees that the final Q-values are proportional to the average reward, and the proportionality factor is $\frac{1}{1-\beta}$. For $\beta \lesssim 1$, 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 and therefore it is crucial to have a look at the actual learning process. The algorithm that we use in order to approximate the steady state (the actual Q-learning algorithm) is the following:

1. Initialize the Q-values.
2. Select a random starting state s which has at least one possible action to select from.
3. Select one of the possible actions. This action will get you to the next state s'.
4. Update the Q-value of the state action pair (s,a) according to the update rule (4) 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

$$Q_{t+1}(s,a) = (1-\alpha)Q_t(s,a) + \alpha[R(s') + \beta \max_{a'} Q_t(s',a')], \tag{4}$$

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

crucial in many applications. Watkins and Dayan (*10*) showed that the Q-learning algorithm converges to the steady state if α converges to zero with certain mathematical properties. However, in our cases we achieved good results with $\alpha = 1$, which is probably because we have deterministic rewards.

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

We use a parameter – the exploration rate $p_{explore}$ – to set the behavior of our Q-learning algorithm. In every step, with a probability of $1 - p_{explore}$ 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. (2) 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. Since this is now a *deterministic* system, the trajectory needs to go to an attractor, which is either a fixed point or a cycle. This is due to the following reason: As the system is discrete and finite, the trajectory eventually needs to come back to a state where it was before. Since the system is deterministic, from then on it will do exactly the same as in the previous cycle.

3 Q-LEARNING FOR DAILY ACTIVITIES OF HUMANS

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

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

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

Now assume once more that the agent has learned some Q-values and now does exploitation only, i.e. it always chooses the action a which maximizes $Q(a,s)$ at any state s.

Because of the time structure, the system cannot go to a fixed point, and so under normal circumstances it will describe a cycle through the 24-hour state space. If the RLP was completely solved, then for $\beta \to 1$ that path maximizes the score (utility) per 24 hours. For smaller β, the situation is similar, but not exactly the same. If the RLP was not completely solved, i.e. some of the Q-values do not correspond to the steady state values, then the agent will nevertheless find a cycle, albeit possibly not the optimal one.

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

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

4 THE "TEST" EXAMPLE

4.1 Description

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

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

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

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

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

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

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

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

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

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

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

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

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

$$\beta_{res_1}^{\left(\frac{t}{t_{res_1}}\right)} = \beta_{res_2}^{\left(\frac{t}{t_{res_2}}\right)}, \tag{5}$$

and then

$$\beta_{res_2} = \beta_{res_1}^{\left(\frac{t_{res_2}}{t_{res_1}}\right)}. \tag{6}$$

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

$$\approx 1 - (1 - \beta_{res_1}) \frac{t_{res_2}}{t_{res_1}}. \tag{7}$$

With $\beta_{res_1} = 0.96$ and $t_{res_1} = 60\,\text{min}$ one obtains

$$\beta_{res_2} = 1 - \frac{0.04 t_{res_2}}{60\,\text{min}}, \tag{8}$$

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

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

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

"High" and "low" are defined with respect to the final Q-values $Q_\infty(s,a)$: A high initial Q-value is larger than any final Q_∞, a low initial Q-value is smaller than any final Q_∞. Some a priori estimates can be obtained from Eq. (3): Q-values are certainly high when they are above $\frac{1}{(1-\beta)R_{max}}$, where R_{max} is the largest reward in the system. Q-values are certainly low when they are below $\frac{1}{(1-\beta)R_{min}}$, 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.

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 $p_{explore} = 0.01$.

4.2 Results

The above scenario was tested with different resolutions. In the coarsest test we used a time resolution of 60 minutes with a discount parameter β of 0.96. In the subsequent tests the resolution was increased by factors of two resulting in 30, and 15 minutes respectively, with corresponding β values of 0.98 and 0.99 according to Eq. (8).

From the design of the utility functions for the four activities "Work", "Shop", "Leisure" and "Home", it follows that the optimal daily plan corresponds to the one shown in table 1(a). 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(a). 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 1(b) we show computational results obtained by our tests. It can be seen that doubling the resolution leads to an increase in the number of iterations needed to converge by a factor of 10. The table gives the minimal number of iterations needed in order to converge with a probability higher than 50%. We considered the algorithm as having converged if both the optimal daily plan corresponded to the one shown in table 1(a) and the agent was able to recover from disturbances in a reasonable manner. The running times were measured on a Mobile Pentium 4 with 2.4 GHz. Our programs were implemented using Java.

One might worry about reliability of the algorithm. As was already mentioned, table 1(b) 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 necessary to increase the number of iterations to 10 times the indicated value or even more. Fortunately, it does not seem to be this way. In fact, we never observed a failure to converge if the algorithm was run for twice the number of iterations stated in the table.

5 A "MORE REALISTIC" EXAMPLE

5.1 Description

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

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

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

Also the travel times between activities were changed, see Tab. 2(a). This was done in order to become comparable with earlier work in our group (*11*).

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

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

With low initialization, we have to make sure that exploration is taken care of by other means. We therefore use higher exploration rates in this case as is also indicated in Tab. 2(d).

Depending on the time resolution we use an exploration rate $p_{explore}$ of 0.4, 0.2 or 0.1 respectively.

5.2 Results

First we ran the Q-learning algorithm multiple times for a long time in order to reliably find a good daily plan for the given reward tables. This was done for each resolution independently. The resulting solutions are shown in Tab. 2(b). Further on, we will refer to these solutions as the optimal solutions.

Similar to the "test" example, we test the algorithm for time resolutions of 60, 30 and 15 minutes respectively and identify the number of iterations needed in order to converge to the best solution with a probability greater than 50%. This we do both for the high initialization approach and the low initialization approach. See table 2(c) for results using the high initialization and table 2(d) for results of the low initialization approach.

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

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

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

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

6 DISCUSSION AND FURTHER WORK

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

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

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

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

After understanding the structure of the problem a bit better, one can in fact envisage much faster algorithms. As becomes clear from the above argument, the slowness of Q-learning lies in the fact that rewards are transmitted against the direction of the algorithm. It turns out that this is in fact only necessary when the expected reward of a state-action-pair is not known (*9*) and the algorithm itself has to do the averaging over the realizations. Since in our case they are known, one can, for example, use faster techniques from Dynamical Programming for the same problem (*12*). Alternatively, one could probably even use a generalized shortest path algorithm (*13*) and simply have it originate at all possible states at a given point in time.

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

As mentioned in the introduction, the problem of activity time selection is “too easy” for Q-learning, since it can also be solved by numerical methods. However, the general

method of expanding the day into possible states along the time axis is amenable to more complicated formulations. For example, one could allow for skipping activities (i.e. have direct transitions to the second-next activity), one could allow for arbitrary activity sequences, and one could include a limited number of different possible locations for each activity. We therefore believe that the general approach – to see daily activity planning as the question to find good cycles through space-time – provides many interesting avenues for future research.

7 SUMMARY

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

The solution found is not only an optimal daily plan but it also holds all information necessary to react to unforeseen disturbances. 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 computational alternatives to “true” within-day replanning.

REFERENCES

1. K. Axhausen et al, editor. *Proceedings of the meeting of the International Association for Travel Behavior Research (IATBR)*, Lucerne, Switzerland, 2003. See www.ivt.baum.ethz.ch/allgemein/iatbr2003.html.
2. *EIRASS workshop on Progress in activity-based analysis*, Maastricht, NL, May 2004.
3. M. Ben-Akiva and S. R. Lerman. *Discrete choice analysis*. The MIT Press, Cambridge, MA, 1985.
4. J.L. Bowman, M. Bradley, Y. Shiftan, T.K. Lawton, and M. Ben-Akiva. Demonstration of an activity-based model for Portland. In *World Transport Research: Selected Proceedings of the 8th World Conference on Transport Research 1998*, volume 3, pages 171–184. Elsevier, Oxford, 1999.
5. R.M. Pendyala. Phased implementation of a multimodal activity-based travel demand modeling system in Florida. See www.dot.state.fl.us/research-center/Completed_PTO.htm, accessed 2004. Project number 0510812 (BA496).
6. C.R. Bhat, J.Y. Guo, S. Srinivasan, and A. Sivakumar. A comprehensive econometric microsimulator for daily activity-travel patterns. *Transportation Research Record*, forthcoming. Also see www.ce.utexas.edu/prof/bhat/FULL_REPORTS.htm.
7. T. Arentze, F. Hofman, H. van Mourik, and H. Timmermans. ALBATROSS: A multi-agent rule-based model of activity pattern decisions. Paper 00-0022, Transportation Research Board Annual Meeting, Washington, D.C., 2000.
8. E.J. Miller and M.J. Roorda. A prototype model of household activity/travel scheduling. *Transportation Research Record*, 1831:114–121, 2003.
9. S.J. Russel and P. Norvig. *Artificial intelligence: a modern approach*. Series in Artificial Intelligence. Prentice Hall, 1995, pp. 598–624.
10. C. J. C. H. Watkins and P. Dayan. Q-learning. *Machine Learning*, 8:279–292, 1992.
11. P. Graf. Artificial intelligence approaches to the learning of humans. Diploma thesis, ETH Zurich, Switzerland, 2003. See e-collection.ethbib.ethz.ch/ecol-pool/dipl/dipl_132.pdf.
12. A. Karlstrom. A dynamic programming approach for the activity generation and scheduling problem. Working paper transport and location analysis, Royal Institute of Technology, Sweden, 2004.
13. C. L. Barrett, R. Jacob, and M. V. Marathe. Formal-language-constrained path problems. *SIAM J COMPUT*, 30(3):809–837, 2000.

LIST OF TABLES

LIST OF FIGURES

TABLE 1 Overview of "Test" Example – (a) Optimal daily plan for the "test" example. – (b) Computational performance with the "test" example using "high" initialization. We show the number of iterations and the running time necessary to converge to the optimal solution, (a), with a probability substantially higher than 50%.

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

(a)

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

(b)

TABLE 2 Overview of "More Realistic" Example – (a) Travel times for the "more realistic" example. Those times are assumed to be constant throughout the day. – (b) Optimal daily plan for the "more realistic" example – (c) Computational performance with the "more realistic" example using "high" initialization. We indicate the number of iterations needed to converge to the optimal solution with a probability substantially higher than 50%. – (d) Computational performance with the "more realistic" example using "low" initialization. We indicate the number of iterations needed to converge to a *reasonable* solution with a probability substantially higher than 50%. $p_{explore}$ is the exploration rate.

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

(a)

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

(b)

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

(c)

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

(d)

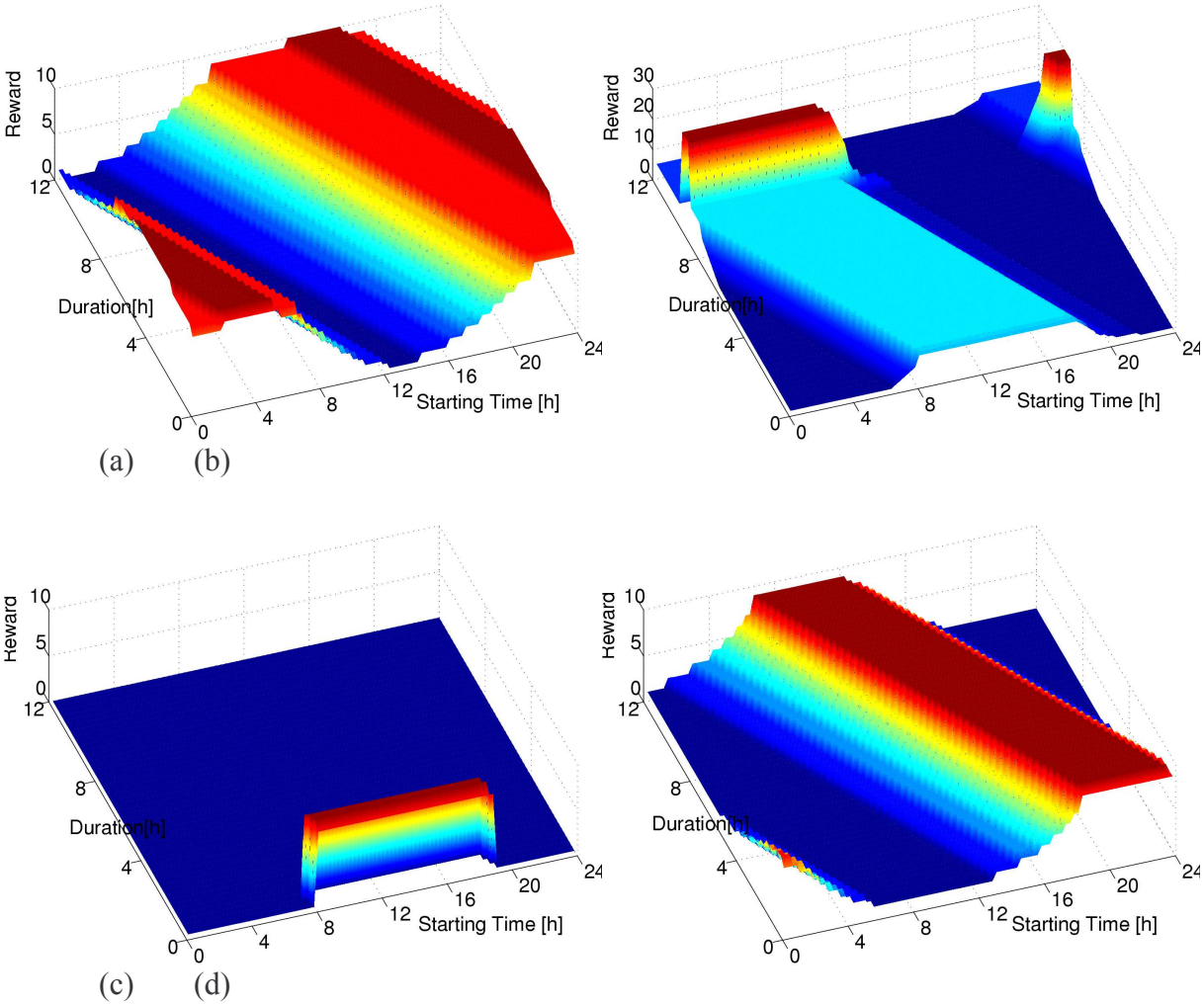

FIGURE 1 Plot of the rewards per 15 minutes for different activities in the “more realistic” example. Rewards depend on the starting time of an activity and the duration that it lasted so far. – (a) Home activity. – (b) Work activity. – (c) Shop activity. – (d) Leisure activity.