Q-learning for flexible learning of daily activity plans

Journal Article**Author(s):**

Charypar, David; Nagel, Kai

Publication date:

2005

Permanent link:

<https://doi.org/10.3929/ethz-b-000023511>

Rights / license:

[In Copyright - Non-Commercial Use Permitted](#)

Originally published in:

Transportation Research Record 1935, <https://doi.org/10.3141/1935-19>

Q-Learning for Flexible Learning of Daily Activity Plans

David Charypar and Kai Nagel

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, $A(s)$, that the agent can take and that 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 stating a sequence of activities (e.g., home-work-shop-home), including locations and times, needs to be found. Activities at different locations generate demand for transportation. Activity generation can be modeled as an 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; that is, the agent can also go on paths through state space in which high rewards are given only at the end. This paper presents computational results with such an algorithm for daily activity planning.

A recent trend in transportation research is to use activities 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, for example, the proceedings of two recent conferences (1, 2). The mainstay of activity-based demand generation is random utility models (RUMs) (3–6). 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 a very large number of options are faced, and (c) they need to be recomputed when a traveler is thrown “off” its optimal day plan. In consequence, alternatives are also investigated, such as behavioral or rule-based approaches [e.g., see Arentze et al. and Miller and Roorda (7, 8)].

This paper investigates whether a certain approach from machine learning, called *Q-learning*, is applicable to activity generation. The two main questions to answer at this stage are (a) Are the results plausible? and (b) Can they be obtained within a reasonable time on

a computer? More expansive research makes sense only if both questions can be answered in the affirmative.

This paper concentrates on a module to generate the time allocation part of activity plans, that is, when activities should begin, how long they should last, and when they should end. The method should, however, also be capable of doing location choice, or activity pattern selection. See the section on discussion and further work for information on this and related methods that could be explored.

This paper begins with a review of *Q-learning*, followed by a section on how to apply it to activity generation. The method is first tested on an unrealistic but somewhat challenging test example and then with a somewhat more realistic case. The paper concludes with a discussion and then a summary.

Q-LEARNING

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 expected discounted reward,

$$E\left(\sum_t \beta^t R_t\right) \quad (1)$$

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 that 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 [see, e.g., Russel and Norvig (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, Equation 1, that can be collected by taking the action.

Q-values are defined in the following way:

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

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

D. Charypar, Institute for Transport Planning and Systems, Wolfgang-Pauli-Str. 15, ETH Hönggerberg, Hil F 121, 8093 Zürich, Switzerland. K. Nagel, Institute for Land and Sea Transport Systems, TU Berlin, Sekr. SG 12, Salzuffer 17-19, D-10587, Berlin, Germany.

Transportation Research Record: Journal of the Transportation Research Board, No. 1935, Transportation Research Board of the National Academies, Washington, D.C., 2005, pp. 163–169.

probabilistic (i.e., several different s' s can result), then some suitable average needs to be taken. β is again the discount parameter in the range $[0, 1]$.

Equation 2, when expanded, leads to a solution of the type $\sum_i \beta^i R_i$. If an agent always takes the action $a \in A(s)$ that 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.

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 $R(s) \equiv \bar{R}(\forall s)$. In that situation, all Q_∞ 's need to be the same, and therefore $\tilde{Q}_\infty = \bar{R} + \beta \tilde{Q}_\infty$, and thus

$$\tilde{Q}_\infty = \frac{1}{1 - \beta} \bar{R} \quad (3)$$

By that argument, it can be seen that the final Q -values are proportional to the average reward, and the proportionality factor is $1/(1 - \beta)$. For $\beta \ll 1$, the factor is very large, and for $\beta \rightarrow 1$, it goes to infinity.

So far, only the steady state of Q -learning has been described, that is, the final solution *after* learning. However, the steady state is not initially known, and therefore it is crucial to examine the actual learning process. The algorithm that is used to approximate the steady state (the actual Q -learning algorithm) is as follows:

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

The update rule is given by the following:

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

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 a parameter of the algorithm. The proper selection of α is crucial in many applications. Watkins and Dayan showed that the Q -learning algorithm converges to the steady state if α converges to zero with certain mathematical properties (10). However, in this case good results were achieved with $\alpha = 1$, which is probably because there are deterministic rewards.

Basically, in each state the agent 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 reaches states that are close to the best solution so far. By that it can refine its knowledge about that solution and collect relatively high rewards. However, 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.

A parameter—the exploration rate p_{explore} —is used to set the behavior of the Q -learning algorithm. In every step, with a probability of $1 - p_{\text{explore}}$, the agent exploits the information stored in the Q -values; with probability p_{explore} , the agent chooses a random action to explore the state space.

Now assume that the agent has learned some Q -values for this situation, either Q_∞ according to Equation 2 or some other values. Assume that now exploration is switched off ($p_{\text{explore}} = 0$), that is, at every state s the agent selects the action a that maximizes $Q(s, a)$. Can anything be said about the long-term behavior in this situation?

This, in fact, describes a discrete dynamical system. Assume also that the number of state action pairs is finite. Because this is now a deterministic system, the trajectory needs to go to an attractor, which is either a fixed point or a cycle. That is due to the following reason: because the system is discrete and finite, the trajectory eventually needs to come back to a state where it was before. Because the system is deterministic, from then on it will do exactly the same as in the previous cycle.

Q-LEARNING FOR DAILY ACTIVITIES OF HUMANS

How can the problem of daily activity planning be encoded in a way that it becomes an RLP? For that, 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 min. Possible states at each time slice are possible activities, for example, home, work, shop. 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 becomes 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 3 p.m., and having arrived at 8 a.m., is different from being at work at 3 p.m., but having arrived at 11 a.m. These rewards are defined most easily with reference to 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, that is, it always chooses the action a that 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-h state space. If the RLP was completely solved, then for $\beta \rightarrow 1$ that path maximizes the score (utility) per 24 h. For smaller β , the situation is similar, but not exactly the same. If the RLP was not completely solved, that is, 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 h. For example, an agent can have one full day in which it wakes up early and goes to bed late, alternated with a less full day in which 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.

TEST EXAMPLE

Description

For testing purposes, the following task was designed: given an activity pattern of four activities—home, work, shop and leisure—the aim is to solve the time allocation problem, that is, find starting and ending times for these four 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.

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

The activities shop and leisure use the same utility function. It is of the same type as the one of home, but the step is at 2 h instead of 7 h. Also the height of the step is set to 2 instead of 7.

The utility function of the 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 h will it 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.

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

These reward tables describe a “difficult” case for the search algorithm, because 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. Generally more plausible reward tables for activities are smoother and give rewards at earlier times in activity execution.

To keep the number of states finite, the maximum duration of any activity is restricted to 12 h. Depending on the resolution, that is equivalent to setting the maximal number of consecutive states spent in the same activity to 12, 24, or 48.

Finally, how the reward tables are derived from the utility functions must be specified. That can be done by calculating the discretized version of the marginal utility function. That 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, a constant travel time between different activities is defined. The choice here is 60 min. The utility of travel is set to be zero. It might be argued that the utility of travel should be negative. However, because all the activities have positive utilities per time, it is already desirable to minimize travel time s .

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

The first problem it would be desirable to solve is to find a reasonable value for the discount parameter β . In principle it would be desirable to choose a value close to one for this parameter because the interest is in finding the day plan that 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, that leads to diverging Q -values.

For efficiency, low discount parameters are best because they reduce interdependency of the Q -values and therefore lead to higher learning speeds. But low discount parameters inherently prefer short activities. That can be to such an extent that long activities (such as work) are left out completely and short activities (such as shop) are repeated over and over again.

Knowing all that, one has to find a compromise. Preliminary tests showed that for time slots of 60 min a discount parameter of $\beta = 0.96$ works well. When the time resolution increases, the length of the activities in regard to number of states also increases and, as a consequence, the discount parameter β has to be increased accordingly. To have comparable results at different time resolutions, it is desirable for the discount per time t to be the same for all times $t > 0$ independent of the time resolution t_{res} . That leads to

$$\beta_{\text{res}_1}^{\left(\frac{t}{t_{\text{res}_1}}\right)} = \beta_{\text{res}_2}^{\left(\frac{t}{t_{\text{res}_2}}\right)} \quad (5)$$

and then

$$B_{\text{res}_2} = \beta_{\text{res}_1}^{\left(\frac{t_{\text{res}_2}}{t_{\text{res}_1}}\right)} \quad (6)$$

For β_{res_1} close to 1.0 and $t_{\text{res}_2} < t_{\text{res}_1}$, this can be approximated by

$$\approx 1 - (1 - \beta_{\text{res}_1}) \frac{t_{\text{res}_2}}{t_{\text{res}_1}} \quad (7)$$

With $\beta_{\text{res}_1} = 0.96$ and $t_{\text{res}_1} = 60$ min, the following is obtained:

$$\beta_{\text{res}_2} = 1 - \frac{0.04 t_{\text{res}_2}}{60 \text{ min}} \quad (8)$$

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

Because the reward tables are completely deterministic 1.0 is chosen as the learning rate α . That 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 because the agent has to find out first for each state that it actually has 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 soon and sticks with it for a long time. As some kind of a compromise, random initialization in a reasonable range is 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 Equation 3: Q -values are certainly high when they are above $1/(1 - \beta)R_{\text{max}}$, where R_{max} is the largest reward in the system. Q -values are certainly low when they are below $1/(1 - \beta)R_{\text{min}}$, 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 selected.

For this problem a “high” initialization with Q -values of 30 is used. 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 must be dealt with is the exploration rate p_{explore} . Because basically exploration is already taken care of by the initialization, the decision was made to use a rather low exploration probability of $p_{\text{explore}} = 0.01$.

Results

The scenario above was tested with different resolutions. In the coarsest test a time resolution of 60 min was used 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 min, respectively, with corresponding β values of 0.98 and 0.99 according to Equation 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. Note that the time between activities is needed for traveling from one location to another.

The solutions found by the algorithm are now compared with the optimal solution shown in Table 1. In addition, the learned Q -values are examined and the ability of the solution to recover from disturbances is appraised. Only if fast and plausible ways of recovering are observed is it assumed that the algorithm has converged to a good solution.

Table 2 shows the computational results obtained by these 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 to converge with a probability higher than 50%. The algorithm is considered as having converged if the optimal daily plan corresponded to the one shown in Table 1 and if 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. The programs were implemented using Java.

The reliability of the algorithm might be a concern. As was already mentioned, Table 2 says something only about the number of iterations needed to converge with a probability of at least 50%. What if a system is needed 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 appear to be that way. In fact, a failure to converge was never observed if the algorithm was run for twice the number of iterations stated in the table.

MORE REALISTIC EXAMPLE

Description

The data used for the test example are not very realistic. The test example was explicitly designed to be difficult to solve: because 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.

TABLE 1 Optimal Day Plan for Test Example

Work	Shop	Leisure	Home
08:00–17:00	18:00–20:00	21:00–23:00	00:00–07:00

TABLE 2 Computational Performance with Test Example Using High Initialization

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

NOTE: Table shows number of iterations and running time necessary to converge to optimal solution, (a), with probability substantially higher than 50%.

However, real life is different. Real activities give rewards at earlier times. For example, being at home pays off 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, new reward tables are introduced for all four activities:

- Activity home (Figure 1a). The reward for spending 1 h at home is independent of the time that was already spent there. The reward depends only 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 (Figure 1b). Work pays off most from 8:00 until 18:00. If the agent performs work outside this time window, the rewards per time slice are gradually reduced. The agent receives a bonus for staying at work for more than 9 h. 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 (Figure 1c). It is assumed that shops are open from 8:00 until 19:00. Therefore, shopping gives rewards only during the day. Maximal shopping time is set to 45 min. If an agent shops for a longer time, it does not receive any reward for the additional time.
- Activity leisure (Figure 1d). The leisure activity is defined similar to the 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.

The travel times between activities were also changed (Table 3). That was done to become comparable with earlier work in the authors' 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 min. All Q -learning parameters were set to the same values as in the test example.

In search of a planning algorithm that is as fast as possible, using an approach with a low initialization was also tried. 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, it is necessary to make sure that exploration is taken care of by other means. Higher exploration rates are therefore used in this case as is also indicated in Table 4. Depending on the time resolution, an exploration rate p_{explore} of 0.4, 0.2, or 0.1 is used.

Results

First the Q -learning algorithm was run multiple times for a long time to reliably find a good daily plan for the given reward tables. That was done for each resolution independently. The resulting solutions

FIGURE 1 Plot of rewards per 15 min for different activities in more realistic example. Rewards depend on starting time and duration of activity: (a) home activity, (b) work activity, (c) shop activity, and (d) leisure activity.

are shown in Table 5. Further on, these solutions will be referred to as the optimal solutions.

Similar to the test example, the algorithm is tested for time resolutions of 60, 30, and 15 min, and the number of iterations needed to converge to the best solution with a probability greater than 50% is identified. That is done for both the high initialization approach and the low initialization approach. See Table 6 for results using the high initialization and Table 4 for results of the low initialization approach.

The number of iterations necessary to converge for the real-world example with high initialization and for the test example are almost the same. That makes sense because 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 is there an observable difference: the real-world example converges to the best solution after 2 million iterations compared with 5 million iterations for the test example.

TABLE 3 Travel Times for More Realistic Example

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

NOTE: Travel times are assumed to be constant throughout the day.

TABLE 4 Computational Performance with More Realistic Example Using Low Initialization

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

NOTE: Number of iterations needed to converge to reasonable solution with probability substantially higher than 50%. p_{explore} is exploration rate.

Compared with the high initialization tests, with low initialization, daily activity plans can be generated much earlier. It appears that usable plans together with reasonable disturbance recovery are produced 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, engineers have the freedom to choose the method that suits their needs better: if speed is more limiting than quality of the solution then they will probably choose the low initialization; otherwise, if the best possible quality is needed high initialization may be the choice.

To picture the meaning of the resulting Q -values, examples of how the agent would recover from disturbances are shown. First, look at what happens if the agent, for some reason, finds itself coming home at 4:00 in the morning. Assuming the 15-min 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 its usual daily plan. As another example, look at what happens if the agent comes to work late. Assume that it starts working at 10:00. Again, looking at the Q -values, the agent decides to stay at work for 8 h (instead of 9.5 h on a normal day) until 18:00 and then change to the next activity, which is shop. Now activity shop begins 30 min later than in the optimal case, namely, at 18:15 instead of 17:45. The agent now decides to spend 30 min shopping and to continue then with activity leisure. Arriving at the next activity at 19:00, the agent is still 30 min late compared with its optimal daily plan. Then, it spends 4 h and 45 min with leisure activities, saving 30 min. 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 escape the undesired situation. In this case, it 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.

DISCUSSION AND FURTHER WORK

It is interesting to know what kind of problem Q -learning is trying to solve. There are $n_{Q\text{-values}} = n_{\text{activities}} \cdot n_{\text{actions}} \cdot \text{size}_{\text{rewardable}}$ Q -values that we are trying to find the steady state for. For the case in which the time resolution t_{res} is 15 min, $n_{Q\text{-values}} = 4 \cdot 2 \cdot 24 \cdot 4 \cdot 12 \cdot 4 = 36,864$.

Therefore any algorithm will need at least 36,864 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. To get a feeling for the problem, consider the following fact:

The 10% time horizon—the number of states a reward has to be away from the current state to affect it by less than 10%—for a discount parameter of $\beta = 0.99$ is $t_{\text{horizon}_{10\%}} = \log_{\beta} 0.1 \approx 229$, 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 it is assumed 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. Because the rewards are transmitted against the direction in which the algorithm proceeds, the algorithm needs to take the whole optimal path/cycle 229 times to have the reward propagated backward to below the 10% level. This argument indicates that an estimation of the number of necessary learning steps is $n_{Q\text{-values}} \cdot t_{\text{horizon}_{10\%}}$, which is $36,864 \cdot 229 = 8.6$ million steps in this case. This is, however, a worst-case argument, which would, for example, 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, the goal is not to find exact Q -values, but only good daily plans. In these tests, the algorithm found good plans with only 2 million visits.

After the structure of the problem is understood a bit better, much faster algorithms can in fact be envisaged. As becomes clear from the argument above, the slowness of Q -learning lies in the fact that rewards are transmitted against the direction of the algorithm. It has been found that this is in fact necessary only when the expected reward of a state–action pair is not known and the algorithm itself has to do the averaging over the realizations (9). Because in this case they are known, it is possible, for example, to use faster techniques than dynamical programming for the same problem (12). Alternatively, even a generalized shortest path algorithm could probably be used (13); it could simply be made to originate at all possible states at a given point in time.

For this implementation of Q -learning, the reward tables need to be filled. These have a large number of values to be defined (for $t_{\text{res}} = 15$ min there are 18,432). The question is where to obtain them from. One option is to use discretized utility functions. For example, the decision could be made to use logarithmic utility functions of the type $U_{\text{act}}(d) = \alpha \log(d/d_0)$, where d is the duration and α and d_0 are parameters. The reward tables could then be filled by setting the reward for the first 15 min to $U_{\text{act}}(d = 15 \text{ min})$, the reward for the second 15 min to $U_{\text{act}}(d = 30 \text{ min}) - U_{\text{act}}(d = 15 \text{ min})$, and so on. The reward tables would retain the advantage that deviations could still be introduced from those mathematical equations when desired, for example, reducing certain rewards for certain times of day. In the authors' 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 because it can also be solved

TABLE 5 Optimal Day Plan for More Realistic Example

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

TABLE 6 Computational Performance with More Realistic Example Using High Initialization

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

NOTE: Number of iterations needed to converge to optimal solution is indicated with probability substantially higher than 50%.

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, skipping activities (i.e., have direct transitions to the second-next activity) could be allowed for, arbitrary activity sequences could be allowed for, and a limited number of different possible locations for each activity could be included. It is therefore believed that the general approach—to see daily activity planning as the question of finding good cycles through space–time—provides many interesting avenues for future research.

SUMMARY

The Q -learning algorithm was used 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, time of day, and 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-h daily plan that can be executed repeatedly on consecutive days.

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

This algorithm was applied to an example with four activities, work, shop, leisure, and home, to generate daily plans of different resolutions. With a time granularity of 15 min, the convergence took no longer than 2 s; for a 30-min temporal resolution, the algorithm had to run for approximately 0.5 s to find the optimum.

Using this method in a multiagent travel simulation to plan the activities of up to 10 million agents appears feasible, and the resulting

within-day replanning capabilities promise computational alternatives to “true” within-day replanning.

REFERENCES

1. Axhausen, K., ed. *Proc., Meeting of the International Association for Travel Behavior Research (IATBR)*, Lucerne, Switzerland, 2003. www.ivt.baum.ethz.ch/allgemein/iatbr2003.html.
2. *EIRASS Workshop on Progress in Activity-Based Analysis*. Maastricht, Netherlands, May 2004.
3. Ben-Akiva, M., and S. R. Lerman. *Discrete Choice Analysis*. MIT Press, Cambridge, Mass., 1985.
4. Bowman, J. L., 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*, Elsevier, Oxford, 1999, pp. 171–184.
5. Pendyala, R. M. Phased Implementation of a Multimodal Activity-Based Travel Demand Modeling System in Florida. Project No. 0510812 (BA496). www.dot.state.fl.us/research-center/Completed_PTO.htm. Accessed 2004.
6. Bhat, C. R., J. Y. Guo, S. Srinivasan, and A. Sivakumar. A Comprehensive Econometric Microsimulator for Daily Activity-Travel Patterns. In *Transportation Research Record: Journal of the Transportation Research Board, No. 1894*, Transportation Research Board of the National Academies, Washington, D.C., 2004, pp. 57–66.
7. Arentze, T., F. Hofman, H. van Mourik, and H. Timmermans. ALBATROSS: A Multiagent Rule-Based Model of Activity Pattern Decisions. In *Transportation Research Record: Journal of the Transportation Research Board, No. 1706*, TRB, National Research Council, Washington, D.C., 2000, pp. 136–144.
8. Miller, E. J., and M. J. Roorda. Prototype Model of Household Activity-Travel Scheduling. In *Transportation Research Record: Journal of the Transportation Research Board, No. 1831*, Transportation Research Board of the National Academies, Washington, D.C., 2003, pp. 114–121.
9. Russel, S. J., and P. Norvig. *Artificial Intelligence: A Modern Approach*. Series in Artificial Intelligence. Prentice Hall, Englewood Cliffs, N.J., 1995, pp. 598–624.
10. Watkins, C. J. C. H., and P. Dayan. Q -learning. *Machine Learning*, Vol. 8, 1992, pp. 279–292.
11. Graf, P. *Artificial Intelligence Approaches to the Learning of Humans*. Diploma thesis. ETH Zurich, Switzerland, 2003. e-collection.ethbib.ethz.ch/ecol-pool/dipl/dipl_132.pdf. Accessed Oct. 5, 2004.
12. Karlstrom, A. A Dynamic Programming Approach for the Activity Generation and Scheduling Problem. Working Paper Transport and Location Analysis. Royal Institute of Technology, Sweden, 2004.
13. Barrett, L., R. Jacob, and M. V. Marathe. Formal-Language-Constrained Path Problems. *SIAM J COMPUT.*, Vol. 30, No. 3, 2000, pp. 809–837.

The Artificial Intelligence and Advanced Computing Applications Committee sponsored publication of this paper.