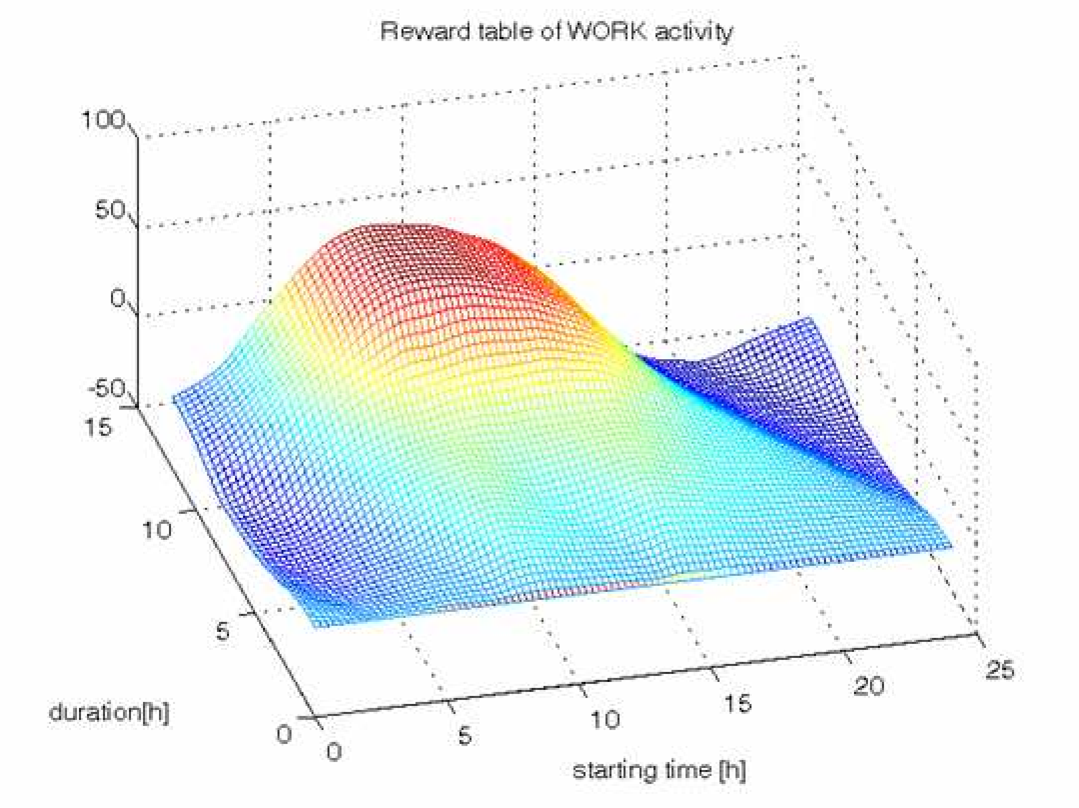

Allocating Time and Location Information to Activity-Travel Patterns through Reinforcement Learning

Janssens Davy, Transportation Research Institute
Lan Yu, Tsinghua University
Wets Geert, Transportation Research Institute
Chen Guoqing, Tsinghua University

Allocating Time and Location Information to Activity-Travel Patterns through Reinforcement Learning

Janssens Davy
Transportation Research Institute
Hasselt University
Belgium
Phone: 0032-11 26 91 28
Fax: 0032-11 26 91 99
E-mail: davy.janssens@uhasselt.be

Yu Lan
School of Economics and Management
Tsinghua University
China

Geert Wets
Transportation Research Institute
Hasselt University
Belgium

Guoqing Chen
School of Economics and Management
Tsinghua University
China

Abstract

Given a sequence of activities and transport modes, for which a framework has been provided in previous work, this paper evaluates the use of a Reinforcement Machine Learning technique. The technique simulates time and location allocation for these predicted sequences and enables the prediction of a more complete and consistent activity pattern. The main contributions of the paper to the current state-of-the art are the allocation of location information in the simulation of activity-travel patterns as well as the application towards realistic empirical data, the non-restriction to a given number of activities and the incorporation of realistic travel times. Furthermore, the time and location allocation problem were treated and integrated simultaneously, which means that the respondents' reward is not only maximized in terms of minimum travel duration, but also simultaneously in terms of optimal time allocation. A computer code has been established to automate the process and has been validated on empirical data.

Keywords

Reinforcement Learning, Q-learning, Activity-based modelling, Agent-based micro-simulation systems, Location allocation, Time allocation

Preferred Citation

Janssens, Davy, Yu Lan, Geert Wets and Guoqing Chen (2006) Allocating Time and Location Information to Activity-Travel Patterns through Reinforcement Learning, paper presented at the 11th International Conference on Travel Behaviour Research, Kyoto, August 2006.

1. Introduction

In the transportation research area, activity and travel modes are critically important information, based on which the transportation demand/status are simulated or predicted. Once the sequential activity-travel combination is known, such as for instance Sleep-Eat-*car*-Work-Eat-Work-*car*-Eat-*bike*-Shop-*bike*-Leisure-*bike*-Sleep, it is meaningful to observe how a learning agent can allocate time and location information for given activity-travel pattern combinations in a reasonable way. Also interesting to observe is how it reacts when it is thrown off its optimal arrangement because of some unforeseeable events, such as for instance a traffic jam. Given a constrained environment, we simulate and look into a learning agent's behavior under the framework of Reinforcement Learning (Mitchell, 1997; Sutton and Barto, 1998), which is in fact a synonym for learning by interaction (Kaelbling *et al.*, 1996). More specifically, the vector <activity, starting time, duration, location> denotes the agent's current state, where duration indicates how long the agent has spent on the current activity. There are two actions available for each state: *Stay* (continue current activity for another time slot at the same location) or *Move* (travel to a possible location where the agent starts to perform the next activity in the pattern). At each state, the agent will receive a reward from the environment when any possible action is chosen. By accumulating this reward information that it obtained from its trial and error search in the state space, the agent finally gets the optimal/satisfactory time and location arrangement. Previous research work in this area generally deals with only one of these two allocation problems: they either focus on the time planning of the activity patterns (Charypar *et al.*, 2004), or search the shortest path in a dynamic programming way (Dijkstra, 1959). In reality, however, a rational person will consider the time and location arrangements simultaneously in order to achieve a total maximal reward. To the best of our knowledge, it is the first time that both problems are integrated and solved using Reinforcement Learning.

Reinforcement Learning goes back to the very first stages of Artificial Intelligence and Machine Learning. As a result of this, the applications of reinforcement learning are situated in the basic roots of artificial intelligence, such as for instance game playing (Littman, 1994; Tesauro, 1992, 1994, Thrun, 1995) and robotics (Mahadevan and Connell, 1992; Schaal, 1994). However, there are also numerous other application domains such as for instance in elevator dispatching (Crites and Barto, 1996), natural systems (Barto and Sutton, 1981), production scheduling (Schneider *et al.*, 1998), but also in a transportation-related context such as in intelligent lane selection (Moriarty, 1998) for achieving a higher traffic throughput. Within an activity-based framework, the reinforcement learning technique has been first applied by Arentze and Timmermans (2003) in the context of learning and adaptation, and only very recently by Charypar *et al*. (2004; 2005) in a time allocation problem.

The remainder of this paper has been organized as follows. The basic conceptions of reinforcement learning are elaborated in section 2, along with the introduction to *Q*-learning, one of the popular algorithms to realize reinforcement learning. In section 3, we will detail by means of artificial examples how *Q*-learning can be applied to the time and location allocation problem respectively, which will help us improve the understanding of reinforcement learning. Section 4 illustrates characteristics and results of activity-travel patterns being optimized in a more realistic environment. Finally, concluding remarks are given in section 5.

2. Reinforcement Learning

Under a constrained environment, the learning agent can perceive a set *S* of distinct states, which are normally characterized by a number of dimensions, and has a set *A* of actions to perform at each state. Reinforcement learning tasks are generally treated in discrete time steps. At each time step *t*, the agent observes the current state s_t and chooses a possible action to perform, which leads to its succeeding state $s_{t+1}=\delta\ (s_t,\ a_t)$. The environment responds by giving the agent a reward $r(s_t,\ a_t)$. These rewards can be positive, zero or negative. It is probable that these preferable rewards come with a delay. In other words, some actions and their consequential state transitions may bring low rewards in short term, while it will lead to state-action pairs later with a much higher reward. On the contrary, an action in a given state may receive an immediate high reward, whereas it makes the agent enter into a path where a series of actions followed, have very low or even negative rewards.

Therefore, the task of the agent is to learn a policy π: $S \rightarrow A$, according to which the agent will achieve the maximal accumulative reward over time. Given an arbitrary policy π from an arbitrary state s_t, the accumulative reward can be formulated as follows:

$$V^{\pi}(s_t) = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \cdots = \sum_{i=0}^{\infty} \gamma^i r_{t+i}$$

where r_{t+i} represents the scalar reward received *i* steps into the future. The constant number $\gamma \in [0,1]$ is the discount factor and it also determines the relative value of immediate versus delayed reward, which indicates how far the agent looks into the future. The agent only consider the immediate reward if γ is set at zero. A paramater γ close to one means that rewards in the far future are given greater emphasis relative to the immediate rewards.

Now the agent is required to learn the optimal policy π^* that maximizes the accumulative reward:

$$\pi^* \equiv \arg\max_{\pi} V^{\pi}(s), (\forall s)$$

We refer to the optimal value function $V^{\pi}*(s)$ as $V^*(s)$ for the sake of simplicity. Given a state *s*, the formula above can be extended with the immediate reward explicitly displayed, which indicates that the optimal action *a* at current state *s* should maximize the immediate reward *r*(*s*, *a*) plus the value *V** of the succeeding state, discounted by γ:

$$\pi^*(s) = \arg\max_a \left[r(s,a) + \gamma V^*(\delta(s,a)) \right]$$

where $\delta(s, a)$ denotes the resulting state after action *a* is performed at state *s*.

2.1 Q-learning algorithm

It is natural to choose *V** as the evaluation function in order to learn the optimal function. Unfortunately, it is required that the perfect knowledge of immediate reward function *r* and state transition function δ are known in advance. When the agent has observed the reward and state transition functions responded by its environment at any state, it is able to calculate the optimal action *a* at any state *s*.

In reality, however, it is usually impossible for the agent to predict in advance the exact outcome of applying an arbitrary action to an arbitrary state. In other words, the domain knowledge is probably not perfect. *Q*-learning (Watkins, 1989; Watkins and Dayan, 1992) is then devised to select optimal actions even when the agent has no knowledge about the reward and state transition functions. It employs the novel evaluation function *Q*(*s*, *a*) as follows:

$$Q(s,a) \equiv r(s,a) + \gamma V^*(\delta(s,a))$$

Then $\pi^*(s)$ and $V^*(s)$ in terms of *Q* (*s*, *a*) can be revised as:

$$\pi^*(s) = \arg\max_a Q(s,a)$$

$$V^*(s) = \max_{a'} Q(s,a')$$

Thus

$$Q(s,a) \equiv r(s,a) + \gamma \max_{a'}(\delta(s,a),a')$$

The recursive definition of *Q*(*s*, *a*) enables *Q*-learning algorithm to iteratively approximate *Q*-values. $\hat{Q}$ is referred as the agent's estimate of the actual function *Q*. The *Q*-learning algorithm maintains a large table with entries to each state-action pair. For each entry, the value of $\hat{Q}$ (*s*,*a*) is stored and initially fulfilled with a random number. The agent repeatedly observes its current state *s*, chooses a possible action *a* to perform, and senses its immediate reward $r(s, a)$ and resulting new state δ (s, a). The $\hat{Q}$ (s,a) value is then updated according to the following rule:

$$\hat{Q}(s,a) \leftarrow r + \gamma \max_{a'} \hat{Q}(s', a')$$

That is to say, the $\hat{Q}$-value of current state-action pair is refined based on its immediate reward and the $\hat{Q}$-value of its next state. After the Q-values of state-action pairs are well estimated by Q-learning algorithm, the agent can choose globally optimal action sequence by repeatedly selecting the action that maximize the local values of Q for current state. The actual learning process can be described as follows (Charypar and Nagel, 2004):

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 leads to the next state s'.
4. Update the Q-value of the state action pair (s, a) according to the update rule above.
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.

2.2 Explore Vs Exploit

The 2nd step in the learning process does not specify how actions are chosen by the learning agent. 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. Because of this, it can refine its knowledge about that solution and collect relatively high rewards. On the other hand, by choosing to explore states that are further apart from the current best solution, it is possible that it discovers a solution that yields higher rewards than the one already known. The strategy above is similar to the local and global search in most known optimization algorithms.

It is common in Q-learning to use a probabilistic approach to selecting actions. One straightforward strategy is ε-greedy method, where the probability of making a random choice is handled by the parameter ε. In every step, with a probability of 1-ε, the agent exploits the information stored in the Q-values, and with probability ε the agent chooses a random action in order to explore the state space.

In the exploration mode, the ε-greedy method assumes equal selection probabilities across possible actions, whereas the chance of selecting a better action may be increased by taking the current value distribution across alternatives into account. A commonly used method assumes a Boltzmann distribution and selects action *a* with probability:

$$\Pr(a_i \mid s) = \frac{e^{\hat{Q}(s,a_i)/\tau}}{\sum_j e^{\hat{Q}(s,a_j)/\tau}}$$

where τ is a parameter usually called the temperature. The higher the temperature, the more evenly probabilities are distributed across alternatives and, hence, the higher the system's tendency to explore (Arentze and Timmermans, 2003). As the temperature decreases, the system assigns increasingly higher probability to the highest valued action, and, hence, the lower the tendency to exploit. The value of the temperature parameter (as well as ε) might be a function of time rather than a constant. Then, the system can simulate a tendency to increase exploration in new environments and decrease this tendency as experience is accumulating.

There are two other possible ways of influencing the system's tendency to explore. First, the choice of the initial value for the value function is relevant. When initial values are set to a high level relative to what can be expected, the system will, even for ε = 0 (in the greedy method) or low temperature (in the Boltzmann method), display a high rate of exploration in an early stage. Second, the system may incrementally update an aspiration level (under each state) and switch to an exploration mode each time the currently best alternative (under the concerned state) drops below the aspiration level.

The reinforcement learning is a Markov decision process (MDP), where the functions δ (*s*, *a*) and *r*(*s*, *a*) depend only on current state and action-pairs. In our application, we will restrict ourselves to a deterministic MDP, for a discussion of a continuous MDP and for more examples, we refer to (Mitchell, 1997). The *Q*-learning algorithm will converge under two conditions. First, the immediate reward is bounded, i.e. there exist some positive constant *c* such that for all state action pair, $|r(s, a)| < c$. The second condition is that the agent selects actions in such a fashion that it visits all possible state-action pairs infinitely often.

3. Time and Location Allocation for Activity and Travel Combinations

In this section, a hypothetical example has been presented to improve the understanding of *Q*-learning. A similar example has been presented and explained in Charypar and Nagel (2004), but is repeated here for the sake of clarity. The behavior of the *Q*-learning algorithm is first explained with respect to the time allocation problem; location allocation is dealt with subsequently. The integration of time and location allocation in a more realistic environment is treated in the next section.

3.1 Time Allocation by means of Q-learning

For this first application and for the sake of clarity, the presence of travel modes has been ignored in the fixed sequence of activities. There are a number of other simplifying assumptions which are made to better understand the behaviour of the agent:

- Fixed order of only 4 activities (1 sequence), i.e.: Home – Work – Shop – Leisure
- Time of the day is discretized with a course time slot of 6 hours. The time structure is assumed to be periodic, i.e. 24:00 P.M. is connected to 0:00 A.M.. The duration of each activity is restricted to 12 hours in order to keep the number of state finite.
- A state *s* is characterized by the activity, starting time of activity and duration (time already spent at activity), and denoted as a triple (*a*, *s*, *d*).
- For a state *s*, an action may be to Stay (‘S’) at the current activity for another time slot or to Move (‘M’) on to perform the next activity.
- No travel time between two activities (ignorance of travel modes)
- Parameter setting: Learning rate α = 1; Discounting factor γ =0.8; ε =1 (ε-greedy method applied).

In addition to these assumptions, reward tables are artificial and extremely simple, as shown in Table 1.

Table 1. An example of a simple reward table for activities

	Home			Work			Shopping			Leisure		
Start time/Duration	0h	6h	12h	0h	6h	12h	0h	6h	12h	0h	6h	12h
0:00 A.M.	0	6	0	0	0	0	0	0	0	0	3	0
6:00 A.M.	0	4	0	0	3	5	0	0	0	0	3	1
12:00 A.M.	0	2	0	0	0	0	0	5	1	0	3	4
6:00 P.M.	0	1	0	0	0	0	0	0	0	0	3	0

It can be seen from Table 1, that the reward of working 0 hours is 0 and is independent of the starting-time of the work-activity. Arriving at work at 6:00 A.M. gives somebody a reward of 3 (units) at the moment he/she is working for 6 hours (i.e. from 6:00 A.M. - 12:00 A.M.) or a reward of 5 (units) at the moment the person is working for 12 hours (i.e. from 6:00 A.M. - 6 P.M.). Arriving at work later than 6 A.M. gives no reward at all. The reward tables for home, shop and leisure are similar.

Let us now reconsider the *Q*-learning algorithm. Since $\alpha = 1$ and $\gamma = 0.8$, the update rule for our simple example is equal to $\hat{Q}(s, a) \leftarrow r(s, a) + 0.8 \max_{a'} \hat{Q}(s', a')$. In the first step of the learning process, all the *Q*-values of every state-action pair are set equal to zero. Next, a random starting state *s* will be chosen, which has at least one possible action to select from. In our example, the starting state may be equal to (Work, 0:00 A.M., 6 hours). The third step selects one of the possible actions, which will bring us to the next state *s'*. Because the exploration probability was set maximal, i.e. ε =1, the agent will always randomly choose an action in order to explore the state space in an attempt to find a new, better solution than the one already known. (On the contrary, when ε =0, the agent will choose the action that has the largest *Q*-value so far.) Suppose the agent randomly chooses to Move on the next activity. The next state turns to be (Shopping, 6:00 A.M., 0 hour). According to the update rule in step 4, the updated *Q* (*s*, *a*) = *Q* (Work, 0:00 A.M., 6 hours; Move) is still equal to 0 since both the immediate reward and the maximal *Q*-value of its next state-action pairs are zeros.

Table 2 shows the states that have been visited by the agent in every loop, while Table 3 illustrates the progress of the *Q*-values for every state-action pair during the execution of the algorithm.

Table 2. Visited states per loop (Numbers denote the loop number)

	Home			Work			Shopping			Leisure		
Start time/ Duration	0 h	6 h	12 h	0 h	6 h	12 h	0 h	6 h	12 h	0 h	6 h	12 h
0:00 A.M.	7	8			1							
6:00 A.M.				9 25	10 26		2	3	4			
12:00 A.M.	13 17			14 18			11 15 19 27			12 16 20 28	21 29	
6:00 P.M.	22 30 34	23	24	31 35			32 (36)			5 33	6	

In the final loop, the state s will be set equal to the state (Shopping, 6:00 A.M., 0 hours). In this artificial example, no travel time has been taken into account. It should be noted that in a realistic scenario, the start time of state s' should thus be augmented with the travel time which is needed to get from state s to state s'. For now, the algorithm continues with loop 2, which starts again at step 3 of the algorithm procedure. The Q-values stay equal to zero until the 5th loop. In this loop, the action is Stay, which will bring the agent to the state (Leisure, 6:00 P.M., 6 hours) and a 3-unit immediate reward. It is worth mentioning that the immediate rewards are given as "utility per time slice", which corresponds to a coarse version of marginal utility. Also interesting to observe is for instance the 23rd loop, where the agent chooses to stay for another 6 hours when it has already been home for 6 hours (start from 6:00 P.M.). The immediate reward is calculated as 0 – 1 = -1, which means that the agent feels unworthy if continues to Stay. The 24th loop is the first where the *Q*-values of its next state-action pairs are non-zeros. The immediate reward is equal to 0, but the second part of the update rule looks at the latest updated *Q*-value for every state-action pair, takes the largest *Q*-value over all the actions and multiplies this by the discounting factor. In this case the latest updated *Q*-value for the state-action pair (Work, 6:00 A.M., 0 hours; Stay) is 3 (see loop number 9) and for (Work, 6:00 A.M., 0 hours; Move) it is 0 (initialization). For this reason, the updated *Q*-value of the 24th loop is equal to = 0 + 0.8 * Max (3, 0) = 2.4. The computation for the other loops is similar (see Table 3).

Table 3. *Q*-values and state-action pairs

Loop	Action	*Q*-value	Loop	Action	*Q*-value	Loop	Action	*Q*-value
1	Move	0	13	Move	0	25	Stay	3
2	Stay	0	14	Move	0	26	Move	0
3	Stay	0	15	Move	0	27	Move	0+0.8 max(3,0)=2.4
4	Move	0	16	Move	0	28	Stay	3
5	Stay	3	17	Move	0	29	Move	0+0.8 max(1,0)=0.8
6	Move	0	18	Move	0	30	Move	0
7	Stay	6	19	Move	0	31	Move	0
8	Move	0	20	Stay	3	32	Move	0+0.8max(3,0)=2.4
9	Stay	3	21	Move	0	33	Move	0+0.8max(1,0)=0.8
10	Move	0	22	Stay	1	34	Move	0
11	Move	0	23	Stay	-1	35	Move	0+0.8max(0,2.4) =1.92
12	Move	0	24	Move	0+0.8 max(3,0) =2.4	…	…	…

The learning procedure continues until each state-action pair has been visited for a sufficient large number of times and until the corresponding *Q*-value converges. Then at each state, the agent chooses the action that achieves a maximal *Q*-value, which means that an optimal policy chart can be constructed as shown in Table 4. Starting from an arbitrary state, the policy will finally guide the agent to its stable and optimal time planning within a day:

Home	:	0:00 A.M.	--	6:00 A.M.
Work	:	6:00 A.M.	--	12:00 A.M.
Shop	:	12:00 A.M.	--	6:00 P.M.
Leisure	:	6:00 P.M.	--	0:00 A.M.

Table 4. Policy Chart for iterations going to infinity

	Home			Work			Shop			Leisure		
Start time/ Duration	0 h	6 h	12 h	0 h	6 h	12 h	0 h	6 h	12 h	0 h	6 h	12 h
0:00 A.M.	S	M	M	M	S	M	M	S	M	M	M	M
6:00 A.M.	S	M	M	S	M	M	S	M	M	M	M	M
12:00 A.M.	M	M	M	M	M	M	S	M	M	S	S	M
6:00 P.M.	M	S	M	M	M	M	M	M	M	S	M	M

For instance, the algorithm will first choose a random start state. Let's say (Shop, 0:00 A.M., 6 hours). The corresponding action in the policy chart is Stay. As a result, the next state is equal to (Shop, 0:00 A.M., 12 hours). According to the policy chart, the agent chooses to Move (since the maximal duration for each activity is 12 hours), and comes to the next states (Leisure, 12:00 A.M., 0 hours). Carrying out the policy in Table 4, the agent sequentially arrives at (Leisure. 12:00 A.M., 6 hours), (Leisure, 12:00 A.M., 12 hours), (Home, 0:00 A.M., 0 hours), (Home, 0:00 A.M., 6 hours), (Work, 6:00 A.M., 0 hours), (Work, 6:00 A.M., 6 hours), (Shop, 12:00 A.M., 0 hours), (Shop, 12:00 A.M., 6 hours), (Leisure, 6:00 PM, 0 hours) and (Leisure, 6:00 PM, 6 hours). Next the agent will Move again to the state (Home, 0:00 A.M., 0 hours), thus forming a cycle within a day, which is the same as the optimal time planning above. It can be seen from the policy chart that an arbitrary start state, such as (Leisure, 0:00 A.M., 6 hours) or (Home, 6:00 A.M., 12 hours), will ultimately lead to the same optimal solution.

Finally, some remarks need to be made with respect to the use of the *Q*-learning algorithm to solve the time allocation problem. First, 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. Second, 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 an activity takes considerably longer than expected, the *Q*-values at the arrival state will still point the way to the best continuation of the plan, as shown in the example above. Third, it is possible that some of the *Q*-values do not converge when their state-action pairs have not been sufficiently visited. Then the agent will nevertheless find a cycle, albeit possibly not the optimal one. In reality, it may be time consuming to visit each state-action pair infinitely in a huge state space with many possible actions, which pushes the agent to a tradeoff between the learning time and solution quality.

3.2 Location Allocation by means of Q-learning

Consistent with the time allocation problem, location allocation can also be solved by means of *Q*-learning. For this purpose, it is assumed that people try to maximize/minimize the reward/cost of its travel in total.

Travel distance may not be an optimal measure for determining the burden of travel because it is plausible in a realistic situation that the distance between location A and location B is shorter than the distance between location A and C, while the travel time may be longer (for instance because of a better road network). Furthermore, it is possible that there is a difference in the transport mode that is used.

Translated into a context of *Q*-learning, the agent learns to find a travel policy that achieves maximal reward/minimal cost. It is assumed that the immediate reward of traveling between two locations depends upon the travel mode, and has a negative correlation with travel time.
Again, consider a simple example with the following simplifying assumptions to better understand the behaviour of the decision agent:

- One activity-travel sequence: Home – *public transport* – work – *walk* – leisure – *walk* – shop – *public transport* – Home.
- A state is characterized by the activity and current location, and is denoted as (a, l).
- For a state, an action is to choose the location where the agent can perform the next activity in sequence. Activities can be carried out in a limited number of locations:

Home	:	Location A
Work	:	Location B
Leisure	:	Location C or D
Shop	:	Location E or F

- Only the rewards that come from travel are learned to be maximized.
- Parameter setting: Learning rate $\alpha = 1$; Discounting factor $\gamma = 0.9$; $\varepsilon=1$ (ε-greedy method applied).

In addition to these assumptions, reward tables are artificial and extremely simple, as shown in Table 5.

Table 5. An example of a simple reward table for travel

	Public transport						Walk					
	A	B	C	D	E	F	A	B	C	D	E	F
A	/	-12	/	/	-14	-16	/	/	/	/	/	/
B	-12	/	/	/	/	/	/	/	-8	-5	/	/
C	/	/	-	/	/	/	/	-8	/	/	-10	-4
D	/	/	/	-	/	/	/	-5	/	/	-6	-6
E	-14	/	/	/	-	/	/	/	-10	-6	/	/
F	-16	/	/	/	/	-	/	/	-4	-6	/	/

Taking these simplifying assumptions into account, Home and Work can only be carried out at location A and B. It is obvious that the agent only has to decide about the location of Leisure and Shop activities, and each of them has two possible choices. The remainder of this section illustrates the learning procedure of the agent.

After all state-action pairs are initialized as zeros, a random state *s* will be chosen. It should be recalled that the state is defined by an activity and an origin location. Assume that the agent first visits state (Work, B). In the third step of the learning procedure, the agent chooses a random action in order to explore the state space in an attempt to find a better solution than the one already know. Let us assume that action (destination) C has been chosen to perform the next activity Leisure. The travel mode lies on the sequence and here is walk. The updated Q (Work, B; C) thereby equals $-8 + 0.9 * \max (Q\ (\text{Leisure, C; E}), Q\ (\text{Leisure, C; F}) = -8$.

Assume that the agent selects to walk to E for Shop when it is at the new state (Leisure, C), *Q* (Leisure, C; E) turns to be -10 + 0.9 * max (*Q* (Shop, E; A)) = -10. As shown in Table 6, the agent visited these states sequentially. These actions at each state and their corresponding updated *Q*-values are demonstrated in Table 6.

Table 6. Visited states per loop

Origin/Activity	Home	Work	Leisure	Shop
A	4, 8, 12			
B		1, 5, 9, 13		
C			2, 10	
D			6, 14	
E				3, 15…
F				7, 11

Table 7. Q-values and State-action pairs

Loop	Action	Q-value	Loop	Action	Q-value
1	C	Q(Work, B; C) = -8	9	C	Q(Work, B; C) = -8
2	E	Q(Leisure, C; E) = -10	10	F	Q(Leisure, C; F) = -28.12
3	A	Q(Shop, E; A) = -14	11	A	Q(Shop, F; A) = -30.85
4	B	Q(Home, A; B) = -12	12	B	Q(Home, A; B) = -16.5
5	D	Q(Work, B; D) = -5	13	D	Q(Work, B; D) = -5
6	F	Q(Leisure, D; F) = -6	14	E	Q(Leisure, D; E) = -18.6
7	A	Q(Shop, F; A) = -26.8	15	A	Q(Shop, E; A) = -28.85
8	B	Q(Home, A; B) = -16.5	…	…	…

The *Q*-values tend to converge when each state-action pair has been visited for a sufficient large number of times. Then at each state, the agent chooses the optimal action that achieves maximal *Q*-value, thus constructing a policy (chart), as shown in Table 8.

Table 8. Policy chart for iterations going to infinity

Origin/activity	Home	Work	Leisure	Shop
A	B			
B		D		
C			F	
D			E	
E				A
F				A

The optimal location allocation for this sample sequence is thus equal to:
Home (A) – *public transport* – Work (B) – *walk* – Leisure (D) – *walk* – Shop (E) – *public transport* – Home (A)…

According to these stored *Q*-values of each state-action pair, the agent know how to react properly back to the optimal path when something unforeseeable happens. For instance, when location D for Leisure is not available today, the agent carries out Leisure at location C instead. Making use of its *Q*-values information about its two choices at location C, the agent wisely selects F as the location for Shopping. Next, it moves back Home at location A and is situated on the optimal path again.

4. Empirical Results

4.1 Optimizing Activity-Travel Pattern Allocations

The previous two sections have independently considered time and location allocation in an artificial environment. In reality, however, the reward function will be more complex, there may exist a more refined time granular; an abundant number of locations may be available for a certain activity, and the distribution of these locations may be more disarrayed. Because of this, it becomes not so straightforward in the planning of time or locations. Furthermore, people will simultaneously take the time and location arrangements into account in order to get a maximal reward in total. It is recalled that the reward of daily activities depends upon the duration as well as start time, people will not simply endeavor to obtain an optimal route for travel, since such a route design may not be perfectly suitable for the time arrangement of daily activities. On the other hand, when people allocate time for activities, they have to consider the flexible travel times since a number of locations are available for the next activity. The time and location arrangements are therefore interacted. We will integrate the two problems under the framework of *Q*-learning in a more realistic environment, which can be described as follows:

- The elements of sequences are limited to four kinds of activities (i.e. Home, Work, Shop and Leisure) and four kinds of travel modes (i.e. walk, bike, car and public transport).
- Time of the day is discretized with a refined time slot of 15 minutes, and the maximal duration of each activity is 12 hours.
- A state *s* is characterized by activity, starting time of activity, time already spent at activity (duration) and the origin location where the activity is performed.
- For a state *s*, an action *a* may be to Stay: keep performing the activity at current location for another time slot, or to Move: move to a possible location where it starts to perform the next activity. The travel mode the agent uses to reach these locations is determined by the sequence.
- The reward functions of these four activities are illustrated in Figure 1-4.

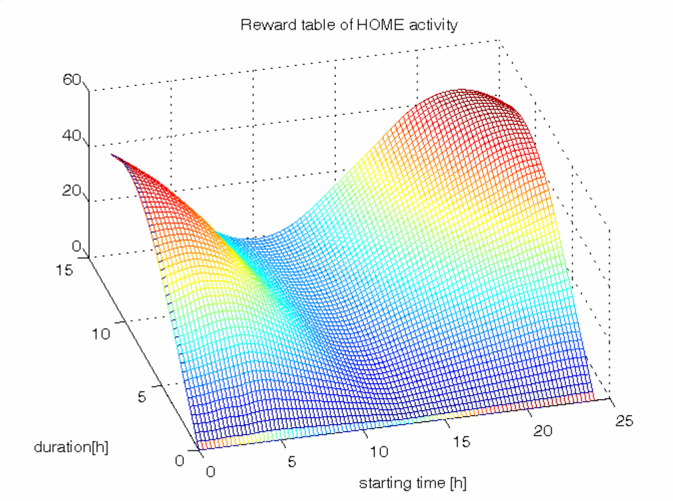

Figure 1. Reward table of Home Activity

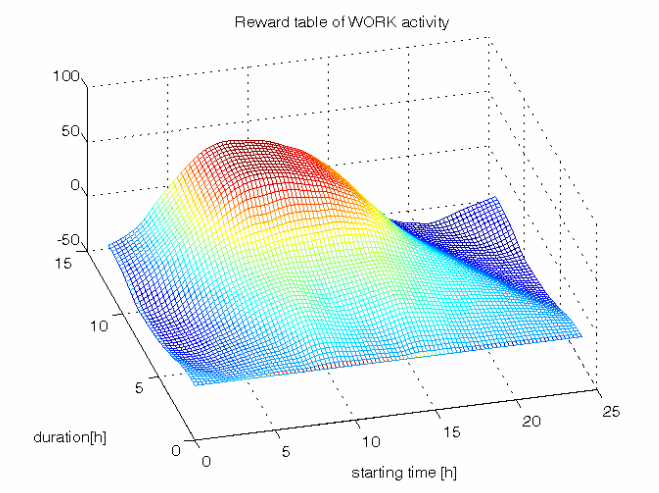

Figure 2. Reward table of Work Activity

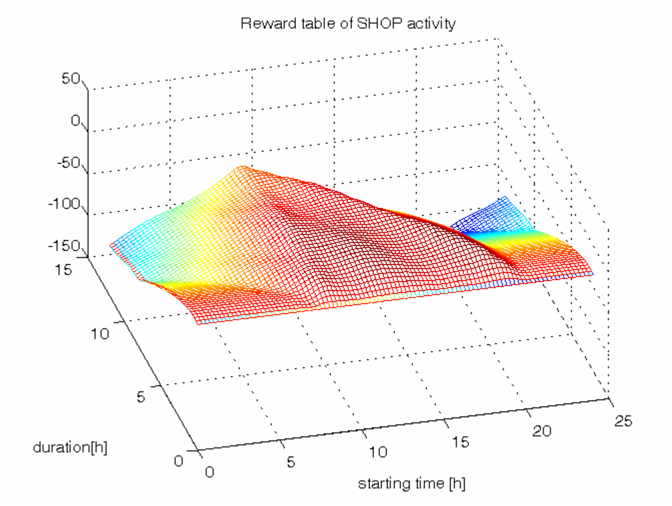

Figure 3. Reward table of Shop Activity

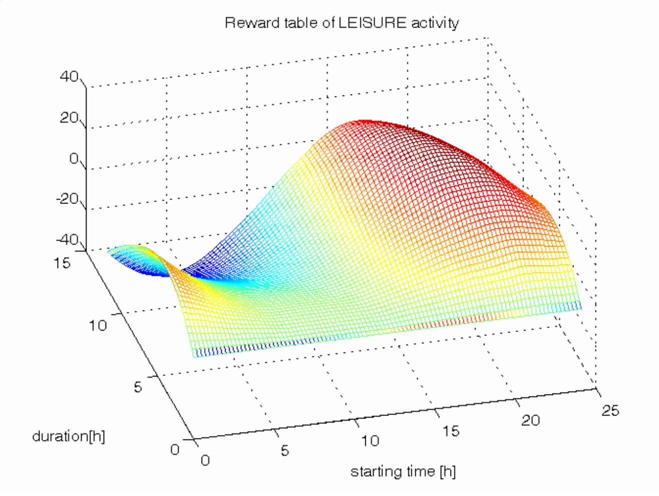

Figure 4. Reward table of Leisure Activity

The rewards of these activities at each starting time and duration combination derive from several factors:

1. For each starting time, the immediate reward returned for a single time slot is determined. For example, starting to work for 15 minutes at 2:00 A.M. brings a negative reward. However, the agent has a much higher reward if it starts to work at 8:00 A.M. for the same duration. Additionally, assuming that the shop is only available between 8:00 A.M. – 8:00 P.M., the agent will acquire no reward if it starts to shop at 6:00 A.M. or continues to shop at 11:00 P.M..
2. The law of diminishing marginal utility is considered. The marginal utility diminished increases nonlinearly as time goes by. However, the extent and trend of the diminishing marginal utilities for each activity are different from each other. For example, the

marginal utility of Shop is decreased much more sharply than that of work after 1 or 2 hours, which make the agent reluctant to keep shopping any longer.

3. The agent will receive a bonus if it keeps working for a reasonable duration, which encourages it to work hard. The amount of bonus also depends on the starting time. The bonus is low if the agent starts to work at 0:00 A.M. even if it works for 8 hours.

Obviously, this is only an experimental setting. As another alternative, an elaborated stated preference experiment can be developed that is able to quantitatively assess the reward that people experience per start time and per time unit that was spent per activity.

With respect to location allocation, 100 locations were collected in a city and we recorded the distances among them. These locations are graphically illustrated in Figure 5, by applying the multi-dimensional scaling (MDS) technique (Johnson and Wichern, 1998). Of these 100 locations, 20 locations are available for Shopping, and 15 for Leisure. For each person, there is only one location available, both for Home and for Work.

For each travel mode, the travel time among these locations are logged. It is assumed that the reward/cost function in term of travel time is as follows:

$$\text{Reward}\left(\text{t}\right) = -c * (b * t)^{a}$$

,where c is identical for all travel modes and is applied to easily control the relative importance of travel compared with daily activities. The parameters b and a are specifically set for each travel mode in order to respectively dominate the range of reward and its evolution trend.

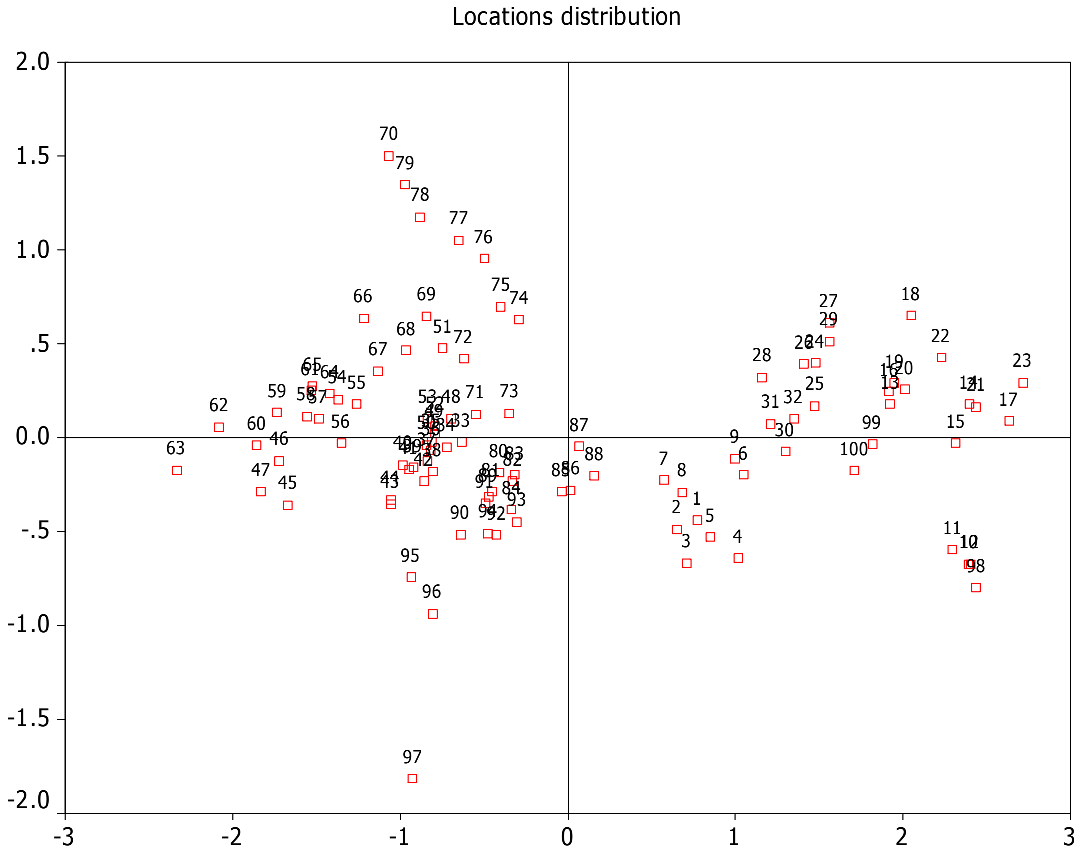

Figure 5. Location distribution

The setting of these parameters are shown in Table 9 and their corresponding curve is depicted in Figure 6.

Table 9. Parameter setting for reward functions of each travel mode

	Walk	Bike	Car	Public transport
a	1.6	1.1	0.6	0.8
b	1/12	1/10	1/6	1/8
c	5			

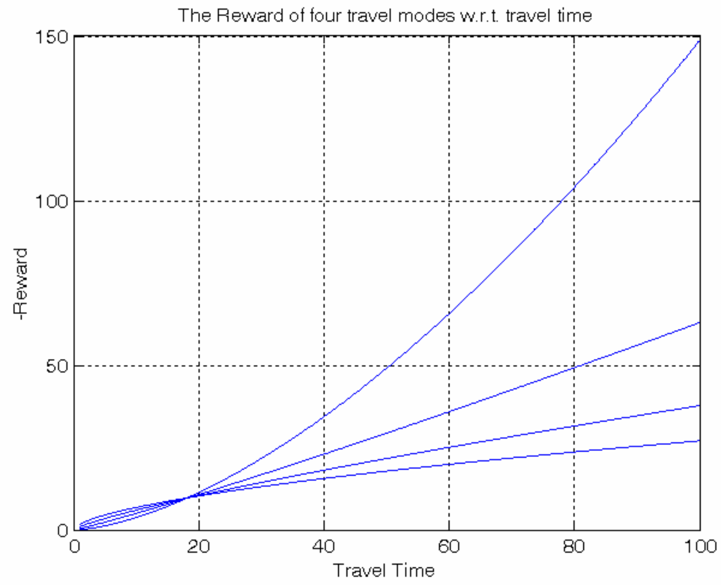

Figure 6. Reward function curves of each travel mode

In such a complicated environment, it is required for the learning agent to look far into the future in order to find a good daily plan of time and locations. The discounting factor γ is set at 0.99, which is close to one and makes the learning procedure harder to converge. The ε-greedy method that was explained before is adopted and ε is set as 1 in order to explore the state space sufficiently.

Due to the use of discrete time intervals, the starting time of activity is calculated as the ending time of previous activity plus, instead of real travel time, the minimal number of time slots that contains the travel time. It is expected that this adaptation causes trivial influence because of the small time granular.

Furthermore, the discount per time slot should be the same during the learning procedure. As a result, the discount factor is equal to γ^m if it takes the agent *m* time slots to travel to the next location.

Three sequences were dealt with in this paper by means of example:

1. Home – *car* – Work – *car* – Shop – *car* – Leisure – *car* – Home
2. Home – *public transport* – Work – *public transport* – Home – *bike* – Leisure – *bike* – Shop – *bike* – Home
3. Home – *public transport* – Work – *walk* – Leisure – *walk* – Shop – *public transport* - Home

The optimal behaviour of three persons are presented for each pattern by means of example. The home and location pair for each person can be listed as follows:

Person A: Home – location 7; Work – location 82
Person B: Home – location 29; Work – location 9
Person C: Home – location 30; Work – location 54

The outputs are displayed in Table 10.

Table 10. Optimal output

Per.	Seq.	Optimal Behavior
A	1	H(23:15 --07:15, 7), W(07:45 --18:00, 82), S(18:15 --19:45, 87), L(20:00 --23:00, 0)
	2	H(22:30 --06:45, 7), W(07:45 --17:30, 82), H(18:30 --18:30, 7), L(18:45 --20:45, 0), S(21:00 --22:00, 6)
	3	H(23:15 --06:45, 7), W(07:45 --17:30, 82), L(18:45 --20:30, 33), S(21:00 --22:00, 36)
B	1	H(23:30 --07:15, 29), W(07:45 --18:00, 9), S(18:15 --19:45, 11), L(20:00 --23:00, 10)
	2	H(22:30 --06:15, 29), W(07:45 --17:15, 9), H(18:45 --18:45, 29), L(19:15 --20:45, 0), S(21:00 --22:00, 3)
	3	H(23:15 --06:30, 29), W(08:00 --17:45, 9), L(18:30 --20:15, 10), S(21:00 --22:00, 11)
C	1	H(23:30 --07:30, 30), W(08:00 --18:00, 54), S(18:15 --19:45, 55), L(20:00 --23:00, 33)
	2	H(22:30 --06:30, 30), W(07:45 --17:30, 54), H(18:45 --18:45, 30), L(19:00 --20:30, 27), S(21:00 –22:00, 25)
	3	H(23:15 --06:45, 30), W(08:00 --18:00, 54), L(18:45 --20:30, 39), S(21:00 --22:00, 38)

*H – Home, W – Work, L – Leisure, S – Shop.

*Each element in the optimal behavior is denoted as Activity (Start time – End time, location).

For example, when person A chooses sequence 2 for everyday life, he/she would like to stay at home from 22:30 P.M. to 6:45 A.M., and then moves by public transport to location 82. At 17:30 PM, he/she stops working and returns home. Person A does not spend in home time (which means that the home activity that was assumed to exist in the given sequence, is skipped) and directly rides bicycle to location 0 for Leisure. After two hours leisure, he heads to location 6 for one hour's shopping. Finally, he starts to move by bike back home at 22:00 P.M.

4.2 Route optimization vs. activity-travel optimization

As mentioned above, the equation Reward (t)= $-c*(b*t)^a$ is applied to calculate the travel reward (cost). We also run our optimization program when c is set as infinite large, which makes the agent arrange his route in a fashion that achieves lowest travel cost. The experiments revealed that for sequences 2 and 3, the route arrangements are the same as those in Table 10, while the situation is probably different for sequence 1. For instance, when *c* is infinite large and sequence 1 is adopted, person A prefer location 83 than location 87 for Shop, and person C prefer location 39 than location 33 for Leisure. The output is the result of the fact that in sequence 1, traveling by car suffers from low cost and the route arrangement is often subject to the activity arrangement in order to achieve highest reward in total, while traveling by public transport, bike or walk is costly and the route should also be carefully designed to alleviate the travel cost as much as possible.

Another interesting output, when *c* is infinitely large, is that the agent will keep performing each activity as long as possible (12 hours in our environment) in order to avoid unnecessary location transfers in everyday life, which is apparently reasonable.

4.3 Back to optimal path gracefully

When unforeseeable events often happen in our life, such as traffic jam or overtime on work, the agent is put off the optimal path. One extraordinary advantage of Q-learning is that the agent, according to the Q-values accumulated in the learning procedure, can wisely choose the appropriate action and move back to optimal path gracefully as it was also mentioned in Charypar and Gärling (2004). We illustrate this characteristic by means of the following two examples.

Example 1: Suppose person A takes sequence 1 as his daily activity-travel pattern. One day, he has to deal with extra tasks and keeps working till P.M. 19:00, which is off his optimal daily arrangement. He then chooses to go shopping at location 87 for 1 hour and 45 minutes. Then he moves to location 0 for leisure and get back home at A.M. 00:00. He will get up at A.M. 07:15 as usual and be on the optimal path again. The adjustment process is as: Work (07:45 --19:00, 82), Shop (19:15 --21:00, 87), Leisure (21:15 --23:45, 0), Home (00:00 --07:15, 7), Work (07:45 --18:00, 82)…

Example 2: Assume person B adopts sequence 2. One morning, he is delayed one hour by traffic jam and starts to work at A.M. 08:45. Based on his experience, he will wisely work for 9 hours. He then arrives at home at P.M. 19:15 and directly moves to location 0 for leisure. After one hour’s leisure, he starts to shop at P.M. 21:00, thus returning to optimal path. The process is stated as: Work (08:45 --17:45, 9), Home (19:15 --19:15, 29), Leisure (19:45 --20:45, 0), Shop (21:00 --22:00, 3), Home (22:30 --06:15, 29), Work (07:45 --17:15, 9)…

5. Conclusion

It is the first time that the activity and route allocations are integrated and optimized in order to achieve maximal rewards in total for a given activity-travel pattern. The reinforcement learning is adopted to help the agent search the optimal path in the huge number of states of given environments.

It is revealed that the activity arrangement and route design are interacted. When the travel cost is small compared with activity reward, the route setting is often subjected to the activity plans and they are not necessary designed by means of the shortest path. The time allocations for activities will also be influenced when the travel cost changes.

Making use of the information obtained from the learning procedure, the agent can wisely choose the optimal action at each possible state. It therefore can gracefully move back to the optimal path when it is thrown off from it by unexpected events.

References

Arentze, T.A. and Timmermans, H.J.P., (2003). Modelling learning and adaptation processes in activity-travel choice: A framework and numerical experiment. *Transportation* **30**, 37-62.

Barto, A.G. and Sutton, R.S. (1981). Associative search network: a reinforcement learning associative memory. *Biological Cybernetics* **40**, 201-211.

Charypar, D., Graf, P., and Nagel, K. (2004). Q-learning for flexible learning of daily activity plans. In Proceedings of the Swiss Transport Research Conference (STRC). Monte Verita, Czechoslovakia.

Charypar, D. and Nagel, K. (2005). Q-learning for flexible learning of daily activity plans. Paper presented at the 84th Annual Meeting of the Transportation Research Board. Washington, D.C., USA.

Crites, R.H. and Barto, A.G. (1996). *Improving elevator performance using reinforcement learning*, in Advances in Neural Information Processing Systems, D.S. Touretzky, M.C. Mozer, and M.E. Hasselmo, Editors. The MIT Press. 1017-1023.

Dijkstra, E. (1959). A note on two problems in connection with graphs. *Numerical Mathematics* **1**, 269-271.

Johnson, R.A. and Wichern, D.W. (1998). *Applied Multivariate Statistical Analysis*. Prentice Hall.

Kaelbling, L.P., Littman, M.L., and Moore, A. (1996). Reinforcement learning: a survey. *Journal of Artificial Intelligence Research* **4**, 237-285.

Littman, M.L. (1994). Markov games as a framework for multi-agent reinforcement learning. Proceedings of the Eleventh International Conference on Machine Learning. San Francisco, CA, USA.

Mahadevan, S. and Connell, J. (1992). Automatic programming of behavior-based robots using reinforcement learning. *Artificial Intelligence* **55**, 311-365.

Mitchell, T. (1997). *Machine Learning* New York: McGraw Hill.

Moriarty, D.a.L., P (1998). Learning cooperative lane selection strategies for highways. Proceedings of the Fifteenth National Conference on Artificial Intelligence. Madison, Wisconsin, USA.

Schaal, S.a.A., C. (1994). Robot juggling: an implementation of memory-based learning. *Control Systems Magazine* **14**, 57-71.

Schneider, J., Boyan, J., and Moore, A. (1998). Value function based production scheduling. Proceedings of the Fifteenth International Conference on Machine Learning. Madison, Wisconsin, USA.

Sutton, R.S. and Barto, A.G. (1998). *Reinforcement Learning: An Introduction*. MIT Press, Cambridge.

Tesauro, G. (1992). Practical issues in temporal difference learning. *Machine Learning* **8,** 257-277.

Tesauro, G. (1994). TD-Gammon, a self-teaching backgammon program, achieves master-level play. *Neural Computation* **6**, 215-219.

Thrun, S. (1995). *Learning to play to game of chess,* in Advances in Neural Information Processing Systems, G. Tesauro, D. Touretzky, and T. Leen, Editors. Morgan Kaufmann, San Francisco.

Watkins, C. (1989). *Learning from Delayed Rewards,* in Psychology Department, University of Cambridge, Cambridge.

Watkins, C. and Dayan, P. (1992). Technical note: Q-learning. *Machine Learning* **8**, 279-292.