Software Agents Coursework 2017

The traveling salesman and the Q-agent

Ronan Greene & Miguel Esteras-Bejar *City, University of London*

"This modest-sounding exercise is in fact one of the most intensely investigated problems in computational mathematics. It has inspired studies by mathematicians, computer scientists, chemists, physicists, psychologists, and a host of nonprofessional researchers. Educators use the TSP to introduce discrete mathematics in elementary, middle, and high schools, as well as in universities and professional schools. The TSP has seen applications in the areas of logistics, genetics, manufacturing, telecommunications, and neuroscience, to name just a few."

An extract from *The Traveling Salesman Problem: A Computational Study* by David L. Applegate, et al., 2006.

Table of Contents

1. Simple scenario

1.1. Define domain and task

The traveling salesman problem (TSP) is defined by a set of cities or locations, and the cost associated to moving from one to another. This cost might represent financial expenses, time, any other cost associated with travelling, or a combination of the above. The salesman (or agent) have to visit all locations and return to his starting location. The learning task involves identifying the order in which the agents needs to visit all the different locations (henceforth called a tour) so that the total cost of the route is minimized. The TSP is often formally represented as a graph problem, where each location is a node, and the edges between these nodes are the possible routes between locations. The following study considers a strict interpretation of the classic TSP; the agent can only visit each node once, and must return to the start node once all other nodes have been visited to complete a tour.

The agent will initially be placed in a relatively simple environment, containing a total of 4 locations fully interconnected. For clarity, this initial environment will be referred to as the 'Toy' problem. The agent will then be placed in more complex environment, inspired by a real case scenario, with a higher number of locations.

1.2. Define a state-transition function

In the context of the TSP, a state is simply a node on the graph, and the available actions are defined by the other nodes (i.e. states) that the agent can travel to from that node. The starting state will always be at the *start* node, and the goal of the agent is to travel to the goal state, which is defined as reaching the *end* node in the graph. We formally define the state-transition function for the TSP environment as $\boldsymbol{s' = t(s, a)}$, where the next state $\boldsymbol{s'}$ that the agent transitions to, is a function of its current state $\boldsymbol{s}$, and the action that it chooses from that state $\boldsymbol{a}$. For each state that an agent occupies, $\boldsymbol{a \in A}$, where $\boldsymbol{A}$ is the set of available actions given $\boldsymbol{s}$. Table 1 demonstrates the state transition function for the set of possible actions available to the agent in the start state of our initial problem.

Table 1: Possible transitions from start state

State (s)	Action (a)	*Next State* ($s' = t(s, a)$)
start	move to state 1	1
start	move to state 2	2
start	move to state 3	3

1.3. Define the reward function

The agent receives an immediate reward each time it transitions between states in the environment. Given the nature of the problem, these rewards are negative values, representing the cost of transitioning between states. In relation to the TSP, this can be thought of as the travel cost between nodes in the graph. The agent will also receive a positively valued goal state reward, for completing a tour. For the initial formulation of the problem, the goal state reward value is set equal to 100. Similar to the state transition function, reward is also a function of the agent's current state, and the action it takes in that state. The reward function is formally defined as $r' = R(s, a)$, where r is the reward received for taking action a in state s. Table 2 demonstrates the reward function for the set of possible actions available to the agent in the start state of the Toy problem.

Table 2: Possible rewards from start state

State (*s*)	Action (*a*)	*Reward* ($r' = r(s, a)$)
start	move to state 1	-10
start	move to state 2	-50
start	move to state 3	-45

1.4. Choose a learning policy

The learning policy is formally defined as π, where the action a that our agent chooses in a given state s, is defined by this policy such that $a = \pi(s)$. Given that the agent starts with no knowledge of the value of actions within the environment, it is desirable that the learning policy have an element of randomness, to allow for exploration. However, it is also desirable that the agent to take into account the value of each action, such that it can learn to exploit the best route. To manage this balance between exploration and exploitation, an ε-greedy policy has been chosen as the default learning policy, where ε represents the exploration factor (Sutton, R.S., Barto, A.G., 2012). Following an ε-greedy policy, the agent will select a random action from the set of all possible actions with probability ε, while with probability $1-\varepsilon$ it will select the action that has the highest value v. The pseudo-code for implementing such a policy is:

1) *generate* x *randomly such that* $x \sim U([0,1])$
2) *if* $x > \varepsilon$ *:*
choose a *randomly from the set* A
else:
$a = argmax\ v(\ t(s, a)\ )$

The ε-greedy learning policy is supplemented using a decay factor, λ. Here the value of ε is updated at the end of each learning epoch such that $\varepsilon' = \varepsilon(1 - \lambda)$. The rationale for this is simple, that the agent explore often early on when it has little knowledge of the environment, and exploit often once it better knows the value of each action.

1.5. Graphical representation

The agent is initially placed in an environment containing 4 cities/nodes, fully interconnected. The relative location of the cities and the travelling cost between them are shown in fig.1. In this case, travelling costs between nodes are symmetrical.

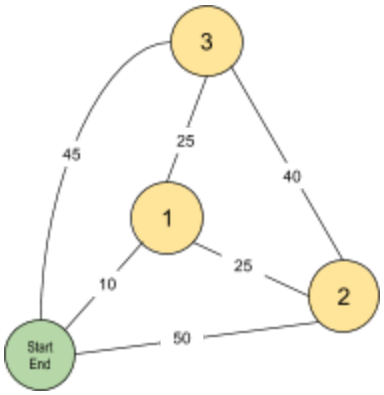


Figure 1. *Graphical representation of the initial TSP scenario.*

1.6. R-matrix

The reward matrix defines the immediate rewards that are available to the agent for all possible actions (a), that it can take from its current state (s). Each row in the R-matrix represents a state the agent can occupy, while each value in a row represents the cost of transitioning from that state, to the state represented by the column header. Blank cells in the matrix define actions that the agent is not allowed to take from a given state. Although the *start* and *end* locations are geographically identical, they are represented as two separate states in the R-matrix. This is just a practical decision to facilitate the learning routine. The reward for transitioning to the *end* includes both the cost of the transition and the goal state reward, as the sum of these two values.

Table 3: R-matrix

	start	**1**	**2**	**3**	**end**
start		*-10*	*-50*	*-45*	
1	*-10*		*-25*	*-25*	*90*
2	*-50*	*-25*		*-40*	*50*
3	*-45*	*-25*	*-40*		*55*

In the matrix represented in table 3, the goal state reward has been defined as 100. Given certain requirements of the TSP, particularly that the agent only visits each node in the graph once, and can only enter the 'end' state once all nodes have been visited, a 'dynamic' R-matrix will be implemented. Again, this is just a practical decision to facilitate the learning routine. Once a state has been entered by the agent, all cells in the column relating to transitioning to this state are updated to be blank, ensuring the agent cannot select an action to return to this state in the future. All cells in the *end* column are also blank until all other nodes in the graph have been visited. At this point they are populated with the appropriate rewards, allowing the agent to transition to this state. This 'dynamic' process is explained in further detail in section 1.8.

1.7. Define & Set the parameter values for Q-learning

The parameters for the Q-Learning process are: Learning Rate (α), Discount Factor (γ), Exploration Factor (ε), Decay Factor (λ) and Learning Policy (π). As the last of these two parameters have yet to be defined, this is done below.

Discount Factor (γ): The discount factor determines the value of future rewards. In Q-Learning, an agent determines the value of a state such that $V(s) = r + \gamma V(s')$, where γ is the discount factor. It is convention for γ to take a value in the range [0, 1]. At the top end of this range (i.e. $\gamma = 1$), the agent values future rewards and current rewards equally. Conversely, with a value of 0, the agent places no value on future rewards, and only takes into account the immediate reward when determining an action. Intermediate values between 0 and 1 accordingly produce intermediate effects between these two extremes.

Learning Rate (α): Q-Learning requires the agent to make estimates of the value of each state in an environment. As the agent explores, these estimates are updated according to additional information received about each state's value. When the agent enters a state, it makes a new estimate of the value of that state (V_{new}), and compares this to it's pre-existing estimate V_{old}. The difference between these estimates can be termed the error, which is denoted as δ, and formally defined for a given state s as $\delta = (r + \gamma V(s')) - V_{old}(s)$. The learning rate controls the degree to which the agent takes into account error estimates when learning about the value of states within the environment, with values for each state being updated such that $V_{new} = \alpha\delta$. Here α is the learning rate parameter, which takes a value in the range [0,1]. Formalised in this manner, it is clear to see the effect of α on the learning process. At the top end of the parameters range (i.e. $\alpha = 1$), a full weighting is attached to new error estimates, and V_{old} is completely replaced with V_{new}. With a value of 0 however, the agent places no value on new error estimates, and V remains unchanged. As with the discount factor, values between these two extremes produce intermediate results.

Prior to running the initial experiments, a default set of starting parameters for Q-Learning are defined (table 4). While there is recognition that this initial selection is somewhat arbitrary in nature, these choices have none the less been made with the intention of selecting a set of starting parameters that would act as a reasonable starting point for our process. Unless explicitly stated otherwise, all experiments run use these default set of values for Q-Learning.

Table 4: Default Q-Learning Parameters

Parameter	Default Setting
α	*0.7*
γ	*0.8*
ε	*1.0*
λ	$5x10^{-4}$
π	ε*-greedy*

In addition to these Q-Learning parameters, there are a number of experimental parameters that also have default settings. These parameters (with default settings in parentheses) are the number of learning epochs (5000) and the number of samples per parameterisation (50). The number of learning epochs is the number of times the agent will run through the TSP environment from start state to goal state. For each combination of parameter settings multiple samples are run, averaging the results in order make the analysis more robust to variance. The number of samples averaged referred to as samples per parameterisation.

1.8 Original Q-matrix and updates during learning

The text below exemplifies the agent learning (Q-matrix update) during one learning episode (epoch).

1. Define initial state (s_0) = start
2. Define possible actions from s_0 (A_0) = 1,2,3
3. Choose an action (a_0=1) from A_0 by ϵ-greedy policy
4. a_0 defines the next state (s_1=1)
5. Possible actions from s_1 (A_1) = 2,3
6. Update $Q[s_0,a_0]$

$$Q[s_0,a_0]new = Q[s_0,a_0]old + \alpha * ((R[s_0,a_0]+...$$

$$\gamma * max(Q[s_1,A_1]) - Q[s_0,a_0]old)$$

$$Q[start,1]new = Q[start,1]old + \alpha * ((R[start,1]+...$$

$$\gamma * max(Q[1,2],Q[1,3]) - Q[start,1]old)$$

$$Q[start,1]new = 0+1*((-10+0.8*max(0,0))-0) = -10$$

7. Choose an action (a_1=2) from A_1 by ϵ-greedy policy
8. a_1 defines the next state (s_2=2)
9. Possible actions from s_2 (A_2) = 3
10. Update $Q[s_1,a_1]$

$$Q[s_1,a_1]new = Q[s_1,a_1]old + \alpha * ((R[s_1,a_1]+...$$

$$\gamma * max(Q[s_2,A_2]) - Q[s_1,a_1]old)$$

$$Q[1,2]new = Q[1,2]old + \alpha * ((R[1,2]+...$$

$$\gamma * max(Q[2,3]) - Q[1,2]old)$$

$$Q[1,2]new = 0+1*((-25+0.8*max(0))-0) = -25$$

R-matrix

	start	1	2	3	end
start		-10	-50	-45	
1	-10		-25	-25	90
2	-50	-25		-40	50
3	-45	-25	-40		55
end					

Q-matrix at t_0

	start	1	2	3	end
start	0	0	0	0	0
1	0	0	0	0	0
2	0	0	0	0	0
3	0	0	0	0	0
end	0	0	0	0	0

Q-matrix at t_1

	start	1	2	3	end
start	0	**-10**	0	0	0
1	0	0	0	0	0
2	0	0	0	0	0
3	0	0	0	0	0
end	0	0	0	0	0

Q-matrix at t_2

	start	1	2	3	end
start	0	**-10**	0	0	0
1	0	0	**-25**	0	0
2	0	0	0	0	0
3	0	0	0	0	0
end	0	0	0	0	0

11. Choose an action (a_2=3) from A_2 by ϵ-greedy policy
12. a_2 defines the next state (s_3=3)
13. Possible actions from s_3 (A_3) = end
14. Update Q[s_2,a_2]

$Q[s_2,a_2]new = Q[s_2,a_2]old + \alpha * ((R[s_2,a_2]+...$

$\gamma * max(Q[s_3,A_3]) - Q[s_2,a_2]old)$

$Q[2,3]new = Q[2,3]old + \alpha * ((R[2,3]+...$

$\gamma * max(Q[3,end]) - Q[2,3]old)$

$Q[2,3]new = 0+1*((-40+0.8*max(0))-0) = -40$

Q-matrix at t_3

	start	1	2	3	end
start	0	**-10**	0	0	0
1	0	0	**-25**	0	0
2	0	0	0	**-40**	0
3	0	0	0	0	0
end	0	0	0	0	0

15. Choose an action (a_3=end) from A_3 by ϵ-greedy policy
16. a_3 defines the next state (s_4=end)
17. Possible actions from s_4 (A_4) = end
18. Update Q[s_3,a_3]

$Q[s_3,a_3]new = Q[s_3,a_3]old + \alpha * ((R[s_3,a_3]+...$

$\gamma * max(Q[s_4,A_4]) - Q[s_3,a_3]old)$

$Q[3,end]new = Q[3,end]old + \alpha * ((R[3,end]+...$

$\gamma * max(Q[end,end]) - Q[3,end]old)$

$Q[3,end]new = 0+1*((55+0.8*max(0))-0) = 55$

Q-matrix at t_4

	start	1	2	3	end
start	0	**-10**	0	0	0
1	0	0	**-25**	0	0
2	0	0	0	**-40**	0
3	0	0	0	0	**55**
end	0	0	0	0	0

1.9. Performance vs episodes

As stated in section 1.7, Q-Learning is run over 100 samples for each selection of parameters experimented with. For each sample, the tour cost in each epoch is recorded, and the mean tour cost across all samples for each epoch is then calculated. It is this sample mean that is then used when discussing the results of each experiment. In presenting these results, the cost of the optimal tour for each TSP problem environment is used as a benchmark for performance. Using this information, Q-Learning performance is measured as the ratio of the cost of the tour taken by the agent in each epoch, to the cost of the optimal tour. It should be noted that as it is not possible for the agent to take a tour that beats the optimal cost, the floor for this measure is 1.0, a score of which signifies that the agent has completed a tour with an optimal route. Using such a measure has the benefit of making the results comparable across different TSP environments. Henceforth, this ratio will simply be referred to as the cost ratio.

Fig. 2 reports the results of applying Q-Learning to the initial Toy problem environment with default parameters. The effect of the agent learning over each epoch is clearly visible, with the cost ratio on average decreasing in each additional epoch, demonstrating that the agent is taking a lower cost route through the environment. The rate of decrease is greatest in the early epochs, levelling off as the agent learns the environment and chooses a greedy policy with a greater frequency. Q-Learning eventually converges to a steady state achieving a near-optimal cost ratio of 1.013, and learning the optimal tour.

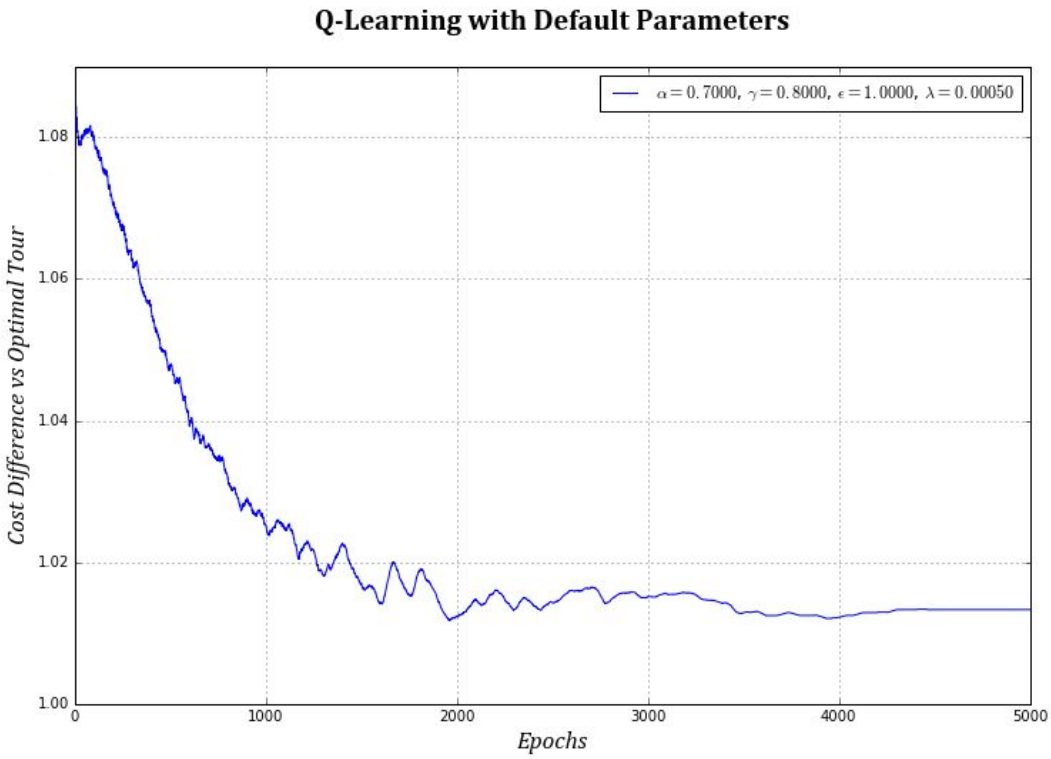


Figure 2. *Results of Q-Learning on Toy problem environment using default parameters*

2. Advanced Cases

2.1. Search of Alpha values

Q-Learning with the default parameters learned the optimal tour within 5000 epochs. In this section alternate learning rates are tested to determine their effect on Q-Learning performance. To better understand the full scope of the relationship between learning rate and performance, a broad range of learning rates are experimented with here. Fig. 3 visualises the results of using α values of 0.001, 0.01, 0.2, 0.6, and 1.0.

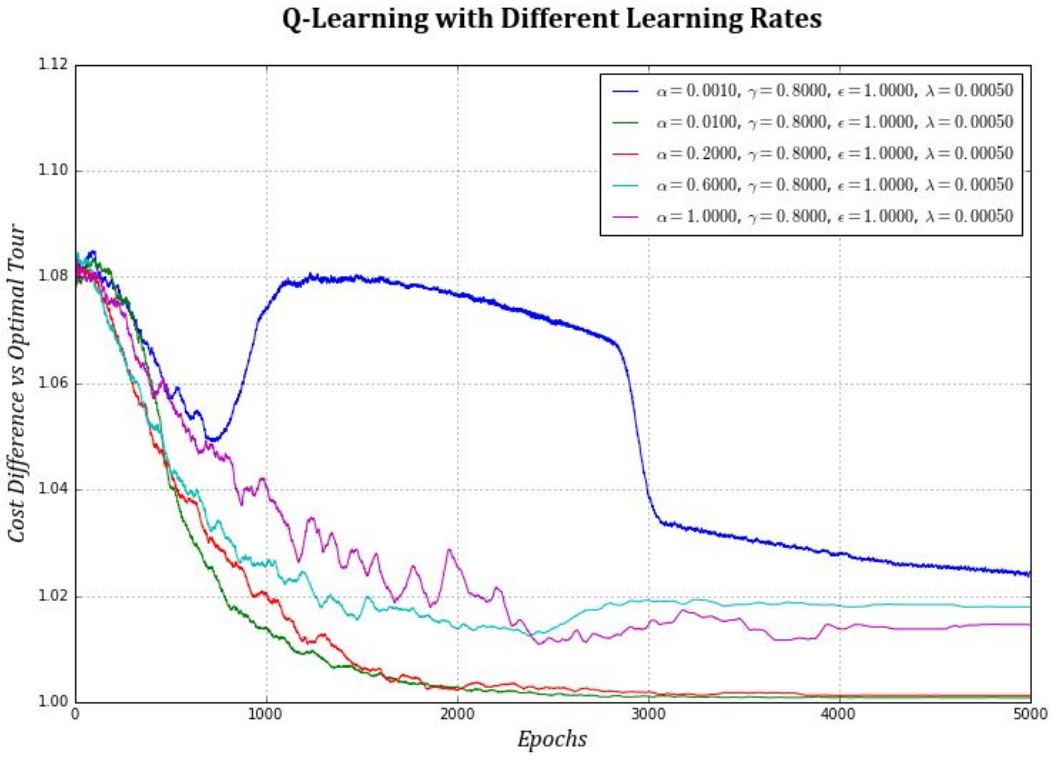


Figure 3. *Results of Q-Learning on Toy problem environment using different learning rates*

The results indicate a clear relationship between learning rate and the cost ratio over each epoch. Using an extremely small learning rate (α = 0.001; blue line) generates the most interesting results, with a rapid rise in cost ratio just before the 1000th epoch, eventually followed by an equally rapid decrease in cost ratio just before the 3000th epoch. It appears that cost ratio is still decreasing at the final epoch, indicating that Q-Learning had yet to converge to a steady-state. Results using alternative values of α are far more predictable, with the cost ratio steadily decreasing and seeming to converge to steady-state prior to the final epoch in each case. Smaller learning rates generally returned better results in terms of the final cost ratio achieved. Learning

rates of 0.010 and 0.200 (green and red lines respectively) performed best in this regard; both converged to a cost ratio of 1.001, virtually optimal. Larger learning rates were clearly less effective, as demonstrated by the plots for α values of 0.600 and 1.000 (cyan and magenta lines respectively). The notable conclusion from this set of experiments, is that an optimal tour cost can be achieved for a simple TSP environment using Q-Learning with an ε-greedy policy, a sufficiently large number of epochs, and an appropriately selected learning rate.

2.2. Search of Gamma values

The next experiment explores the effect of altering the discount factor. With the exception of the γ parameter, that is the subject of this experiment, all other parameters are set to the default values defined in section 1.7. Again, a broad range of discount factors are used to understand the full scope of effect this parameter has. Figure 4 visualises the results of using the following γ values: 0.001, 0.01, 0.2, 0.6, and 1.0

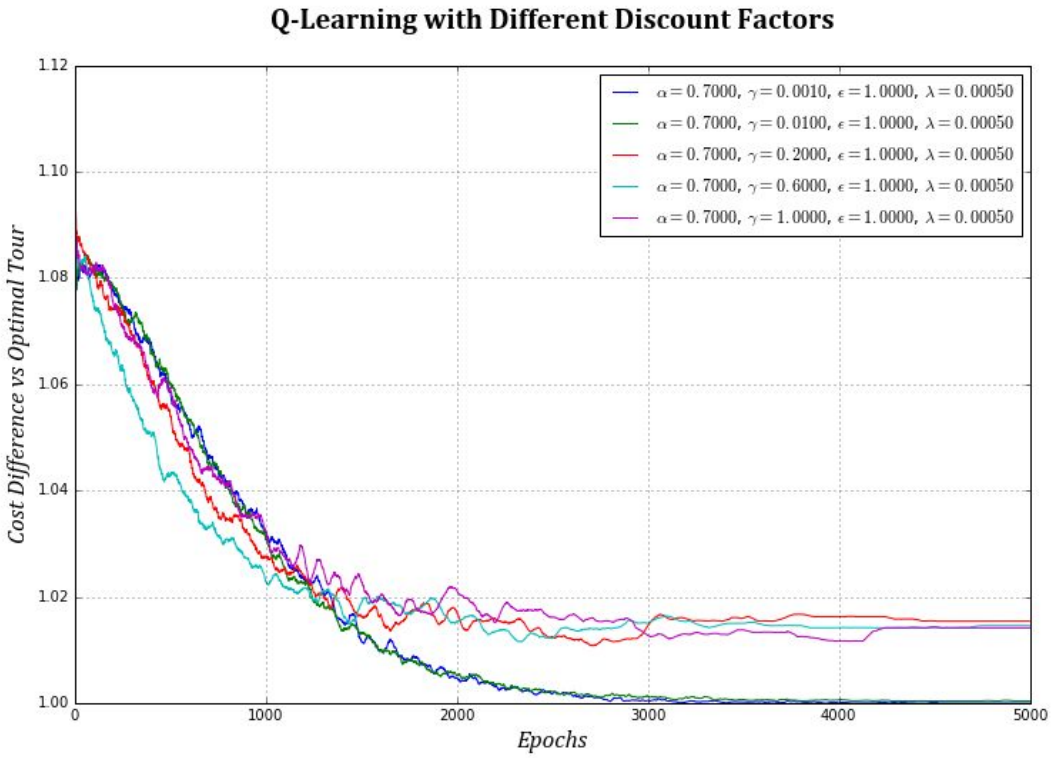


Figure 4. *Results of Q-Learning on Toy problem environment using different discount factors*

The results plotted in figure 4 suggests a fairly consistent relationship between the discount factor and performance. Similar to learning rate, smaller discount factors yielded the best results in terms of cost ratio. Using a γ of 0.001 and 0.01 (blue and green lines respectively) both resulted in Q-Learning converging to a cost ratio of 1.000, a result of the agent learning an optimal solution. Higher γ values achieved respectable but inferior results; gamma values of 0.200, 0.600, and 1.000 all converged to cost ratios in the range of 1.014 to 1.015.

2.3. Search of Epsilon and Epsilon decay values

The final variable in the Q-Learning process relates to the probability with which the agent explores the environment. Two parameters influence this throughout the process. The exploration factor (ε) determines the initial probability with which the agent chooses an exploratory learning policy; the decay factor (λ) controls the rate at which ε decays after each epoch. Throughout these experiments, the decision is made to treat the exploration factor as a constant, with ε fixed equal to 1.00. Agent exploration is then influenced solely by altering the decay factor, with higher values of λ resulting in ε decreasing at a faster rate as the agent learns.

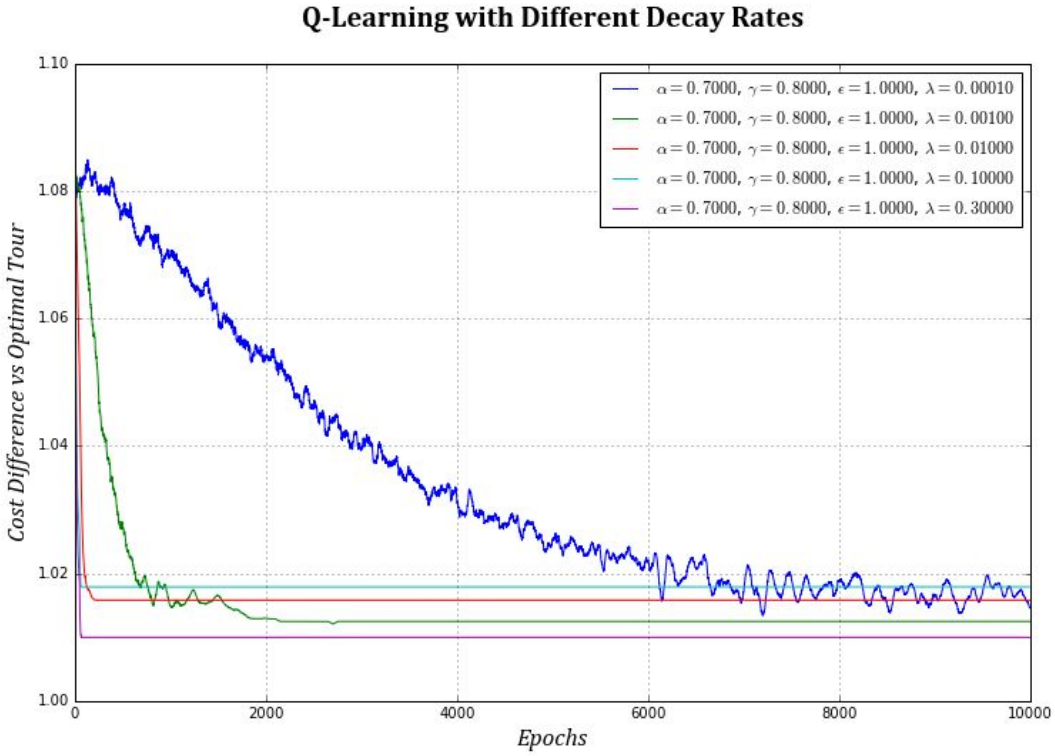


Figure 5. *Results of Q-Learning on Toy problem environment using different decay rates*

Figure 5 reports the results of Q learning with the following values of λ: 1.0×10^{-4}, 1.0×10^{-3}, 0.01, 0.10, and 0.30. Varying λ across this range leads to some quite dramatic Q-Learning results. In particular, larger decay rates result in far sharper decreases in cost ratio over the early epochs, and an earlier convergence to a steady-state. This makes sense intuitively. Small values of λ mean that the agent will favour an exploratory learning policy for a longer period. Large values of λ result in the opposite; the agent starts exploiting its knowledge of the environment more frequently at an early stage. Provided the agent has learned a Q matrix that provides a good approximation of the values of each state by this stage, the cost ratio should naturally reduce significantly once exploitation increases in frequency. Using the smallest decay factor (λ = 1.0×10^{-4}; blue line) resulted in the slowest decrease in cost ratio. Even over 10,000 epochs,

Q-Learning using this parameterisation failed to converge; most likely as the agent was still choosing an exploratory policy with a significant probability at this point. In contrast, Q-Learning with higher values of λ resulted in sharp decreases in cost ratio early on. To better visualise the results of these parameterisations, fig. 6 shows the results from the same experiments over a limited range of epochs.

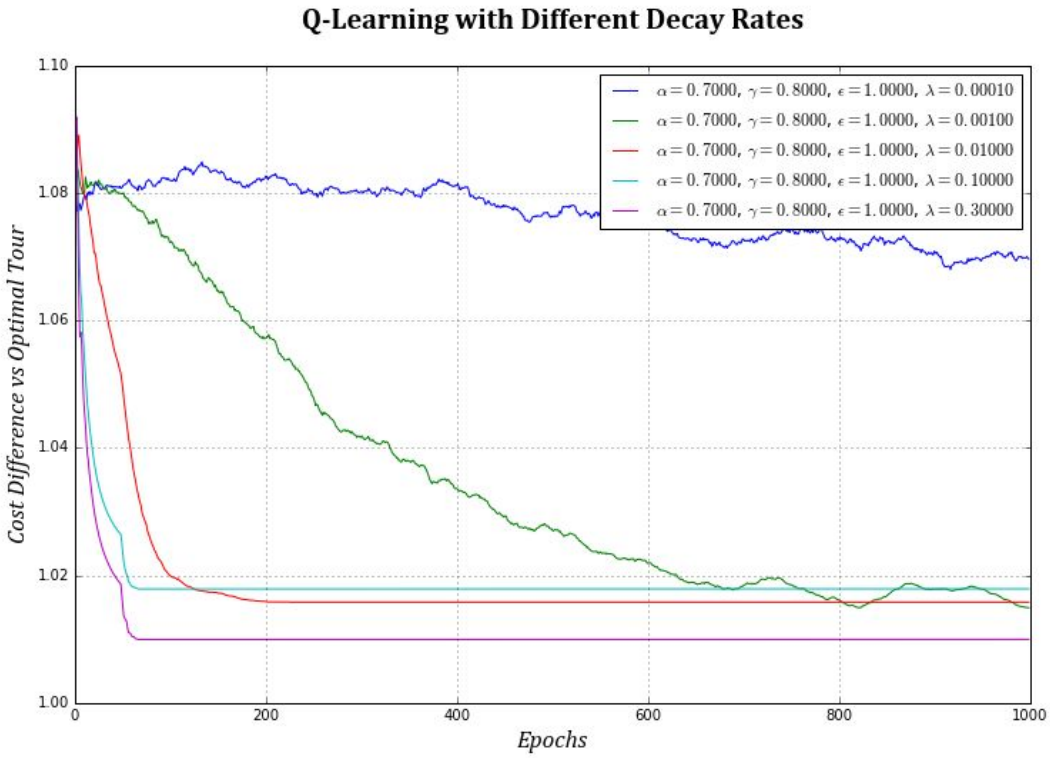


Figure 6. *Results from fig. 5 reproduced, with the x axis limited to the first 1000 epochs*

Interestingly, all parameterisations of λ tested here failed to result in the agent learning the optimal cost tour route. This is in contrast to the experiments with learning rates and exploration factors, which both demonstrated that an optimal solution could be achieved by altering these parameters. The relationship between decay rate and cost ratio at convergence is unclear, with cost ratio fluctuating as decay rate is increased across the sample of values experimented with. This leads to the conjecture that this is a complex relationship, reliant on the subtleties of the environment being explored, and the resulting Q matrix that has been generated once ε has become small enough to be insignificant.

2.4. Searching an optimal combination of parameters

The experiments performed in the sections above demonstrated that, via manipulation of the α and γ parameters, Q-Learning is capable of achieving the optimal cost ratio the Toy problem

environment. Additionally, manipulation of λ values have produced sub-optimal but rapidly converging solutions. The primary objective of applying Q-Learning to the TSP is that of the agent learning an optimal cost tour route. However, a secondary but also important objective is to achieve this in the smallest possible number of epochs. This section reports the results of a search for parameters that maximise performance in this regard. That is, to achieve a least cost solution in the least possible number of epochs. With this aim, a simple parameter search strategy is implemented:

1. Start with the combination of parameters that achieved the best results in section 2.3
2. With all other parameters fixed, vary alpha and retain the best performing value
3. With all other parameters fixed, vary gamma and retain the best performing value

In step 2 of this strategy, a range of learning rates are used in combination with γ and λ parameters fixed at 0.80 and 0.30 respectively (fig.7).

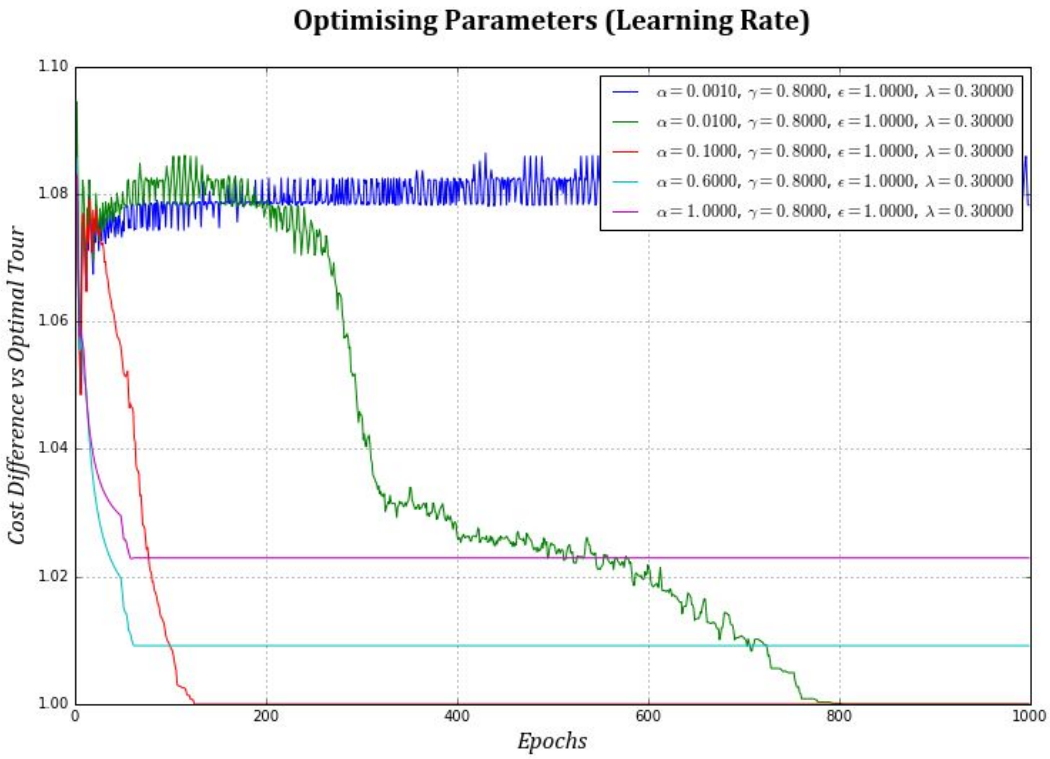


Figure 7. *Results of Q-Learning with best decay rate, default discount factor and different learning rates.*

Similar to the results reported in section 2.1, larger learning rates converge to sub-optimal cost ratios, while very small learning rates fail to converge over the finite number of epochs experimented with. The significant difference in this context is that the larger decay rate has greatly accelerated the number of epochs taken until convergence. Learning rates values of 0.01 and 0.10 both result in optimal solutions in this instance, however performance of the later is clearly superior in terms of the secondary objective; convergence occurred at epoch 125 .

The optimal parameter search proceeds with retaining 0.10 as the best performing learning rate, and testing this in combination with a range of discount rates. The results of these tests are shown in fig. 8. All variants of these parameterisations resulted in a the agent learning an optimal cost tour in 365 epochs or less. A very clear relationship between γ and epochs to convergence was exhibited for the range of parameters tested; larger discount factors unequivocally resulted in

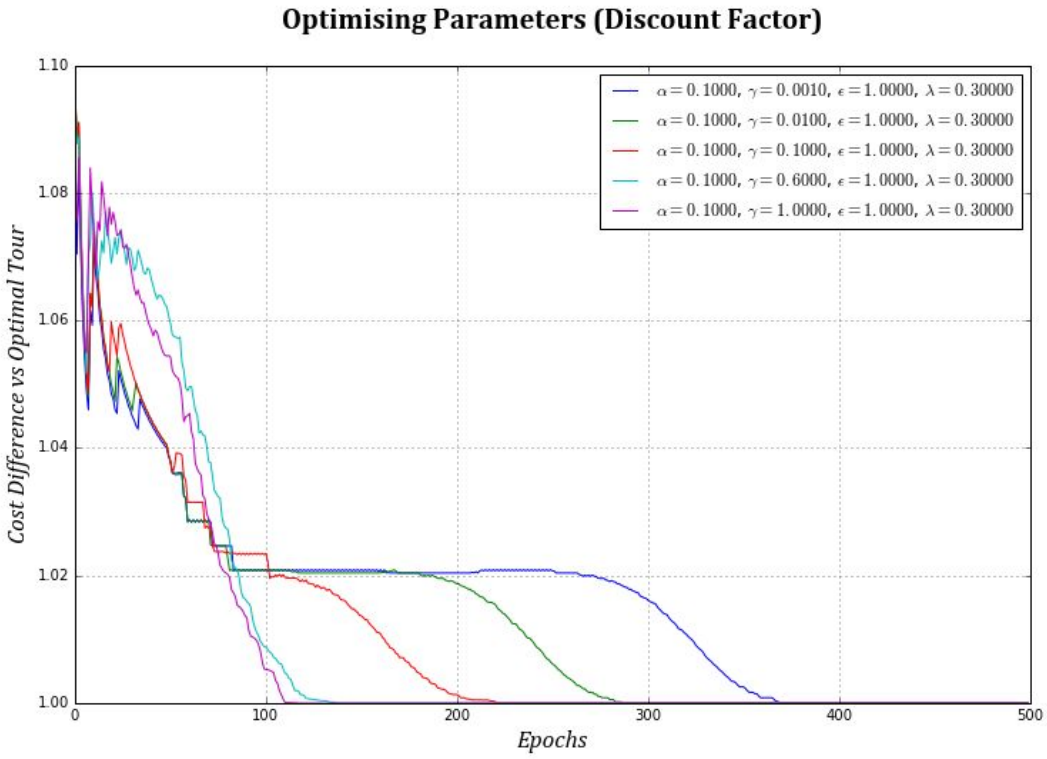
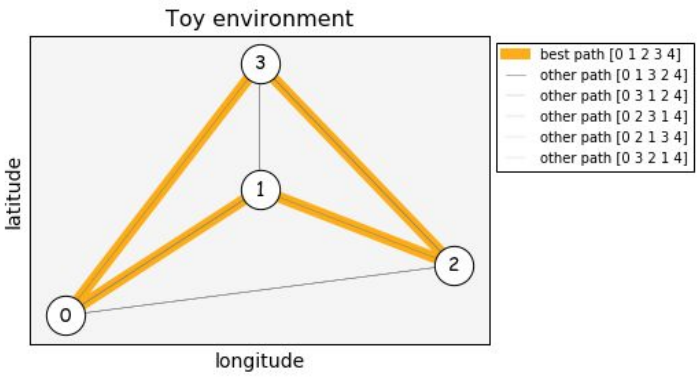


Fig 8. *Results of Q-Learning pairing best decay rate and learning rate with different discount factors*

faster convergence. As such the best value of γ was 1.00, resulting in the agent learning an optimal tour in just 109 epochs. The full set of parameters used to obtain this performance are defined as: α = 0.10, γ = 1.00, ε = 1.00, λ = 0.30. Fig. 9 graphs a representation of tours taken by the agent while learning with this set of parameters.

Figure 9. *Graphical representation of tours taken by the agent during training. The minimum cost tour learned by the agent is represented in orange, all other paths taken during learning are in grey. The start/end node have the same location and are represented with values 0 and 4 respectively.*



2.5. Using different reward functions

The final experiment using the Toy TSP environment takes the optimal parameter set defined in section 2.4 and tests this parameterisation with altered goal rewards. The objective is to determine the impact that the goal reward has on the agent's ability to learn in the environment. The results of this experiment demonstrate that setting of the goal reward is not insignificant in this scenario (fig.10).

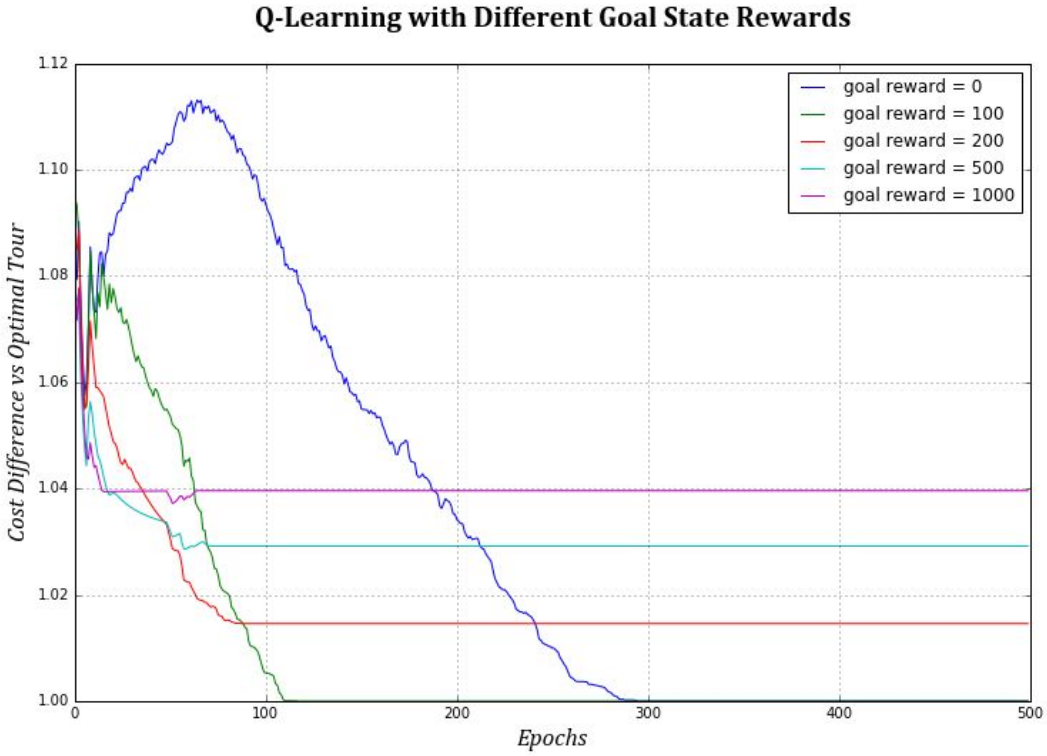


Figure 10. *Results of Q-Learning using different goal state rewards with best set of learning parameters*

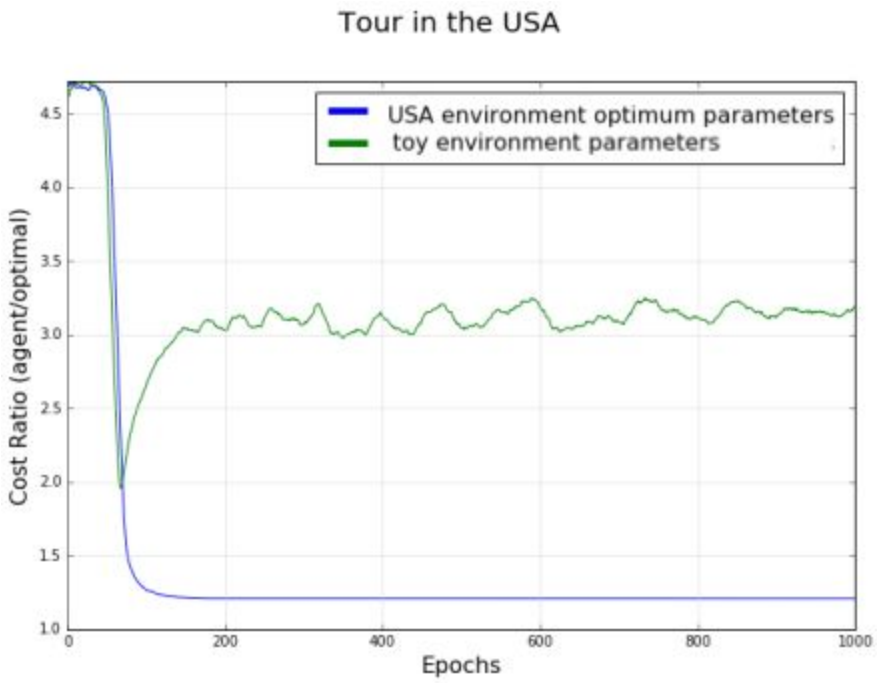
All goal rewards greater than the default value resulted in Q-Learning converging to a sub-optimal solution. Of course, this is based on a set of parameters that were selected based on their performance with a goal state reward of 100. It is plausible to assume that these higher goal state rewards could achieve the optimal cost ratio with different parameterisations. Interestingly, the agent managed to learn an optimal tour route using a zero goal state reward, albeit with a longer time to convergence. In retrospect this is not incredibly surprising, the environment is highly deterministic in that the agent will always be forced to complete a tour in exactly five state transitions. Another way of framing this to say that no final reward is required to incentivise the agent to transition into the goal state, as this will happen as a matter of due course.

2.6. Complex environment

In this section, the same strategy to solve the TSP with Q-Learning will be tested on a different scale. Given the good performance of the model given a simple environment, it is expected that the agent will be able to approximate the optimum tour given an arbitrary number of states/cities. The new environment represents the map of the USA., containing 48 cities.

As in the previous task, the salesman (agent) needs to learning a tour that minimizes the travelling cost while visiting all cities and returning to the starting point. All cities are fully interconnected and the travelling cost between them is calculated based on the geographical distance only. The optimal tour around the USA that minimizes the travelling cost (optimal cost = 33551) is shown in figure 12B.

In this case, the goal state reward value is set equal to 1000. Smaller goal state rewards were used with no observable difference in the results. The experiments show that the agent is able to approximate the optimal tour by Q-learning but never reach it (minimum agent cost = 40551). Q-learning hyperparameters search (data not included) shows that all best models converge to the same minimum cost and tour (fig. 11&12).



Toy env. Parameters:
epochs = 1000, alpha = 0.1, gamma = 1, epsilon = 1, epsilon_decay = 0.3

USA env. parameters:
epochs = 1000, alpha = 0.9, gamma = 0.01, epsilon = 1, epsilon_decay = 0.001

Figure 11. *Performance of Q-Learning in the USA environment. The toy env. parameters are those that show optimum results in the toy environment (described in previous section). USA env. parameters (alpha, gamma and epsilon_decay) were determined by hyperparameters search as seen done for the toy environment.*

A

Optimal tour compared with Agent's best tour

start(0), 7, 37, 30, 43, 17, 6, 27, 5, 36, 18, 26, 16, 42, 29, 35, 45, 32, 19, 46, 20, 31, 38, 47, 4, 41, 23, 9, 44, 34, 3, 25, 1, 28, 33, 40, 15, 21, 2, 22, 13, 24, 12, 10, 11, 14, 39, 8, end(0).

start(0), 8, 37, 30, 43, 17, 6, 27, 5, 36, 18, 26, 42, 29, 35, 45, 32, 14, 11, 10, 22, 13, 24, 12, 20, 46, 19, 39, 21, 15, 2, 33, 4, 47, 38, 31, 23, 9, 25, 3, 34, 44, 41, 28, 1, 40, 7, 16, end(0).

B

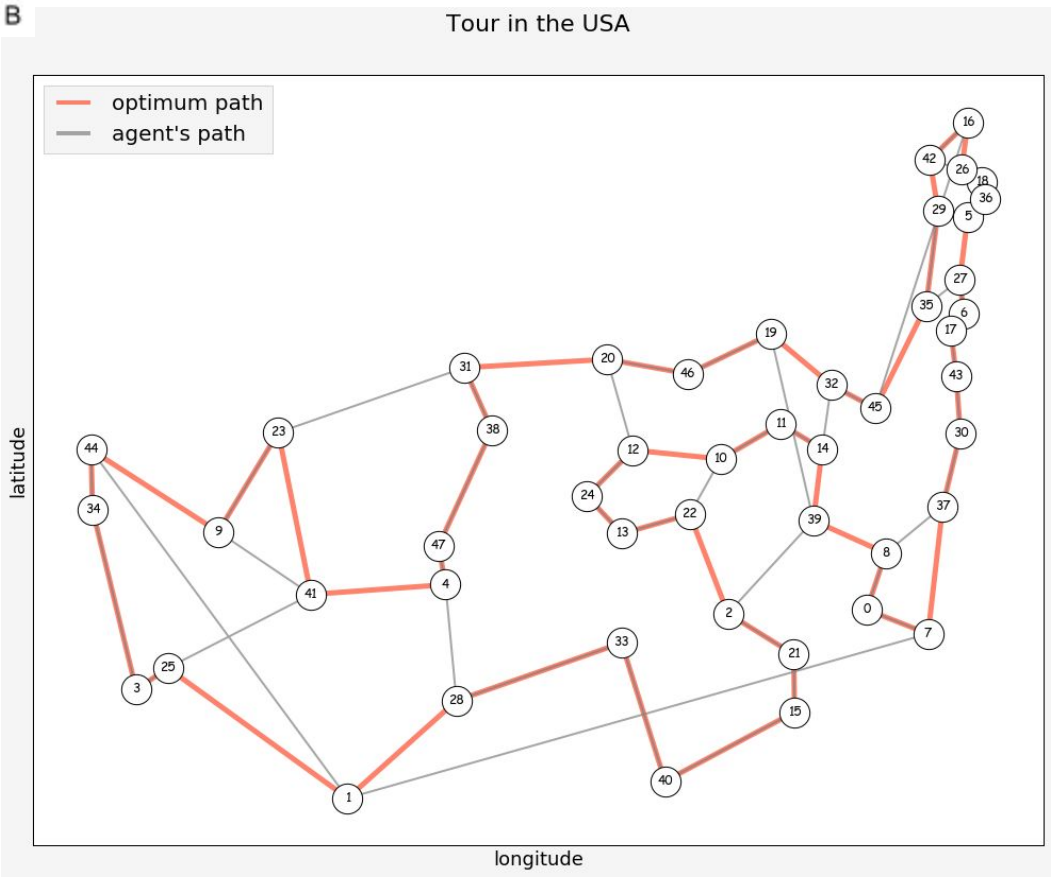


Figure 12. *Comparison between the optimal tour (minimum travelling cost) around the 48 cities in the USA and the minimum cost tour learned by the agent. A describes the minimum cost tour learned by the agent compared with the optimal tour. B shows the tour map. The real longitude and latitude values of the cities are used to represent their location. Learning parameters: epochs = 500, alpha = 0.9, gamma = 0.01, epsilon = 1, epsilon_decay = 0.001.*

2.7. Double Q-learning

Q-learning as described in section 1.8 uses the same values in the Q-matrix for choosing actions (by ε-greedy policy) and for the estimation of future rewards (maximum policy). As a consequence, the algorithm adds a positive bias on the model (or maximization bias), which consistently overestimates future rewards. Using different values to choose the actions and for estimations eliminates this bias and can improve the performance of Q-learning under some conditions (Hasselt, H. Van, 2010). This variation of the Q-learning algorithm, named Double Q-Learning, was recently used as part of a deep-reinforcement learning strategy to improve the performance of agents playing some video games (Hasselt, H. Van, et al., 2015).

In the case of the TSP, Double Q-Learning will be implemented during the learning process in the U.S.A. scenario (experimental results shown in fig. 13&14). The agent will learning two separate Q matrices (Q1 and Q2). At every time step, there are two possible Q-matrix update paths (with equal probability);

- ½ probability of: using **Q1** to choose an action (ε-greedy policy), using **Q2** to estimate its value.

$$Q1[s_t,a_t]new = Q1[s_t,a_t]old + \alpha*((R[s_t,a_t] + \gamma*Q2[s_{t+1},argmax(Q1[s_{t+1},a]] - Q1[s_t,a_t]old)$$

For every time step two estimates, from **Q1** and **Q2**, are calculated, however, only the **Q1** estimate is updated. Hence there is a double memory requirement but not increased in computation.

- ½ probability of: as above, with **Q1** and **Q2** switched.

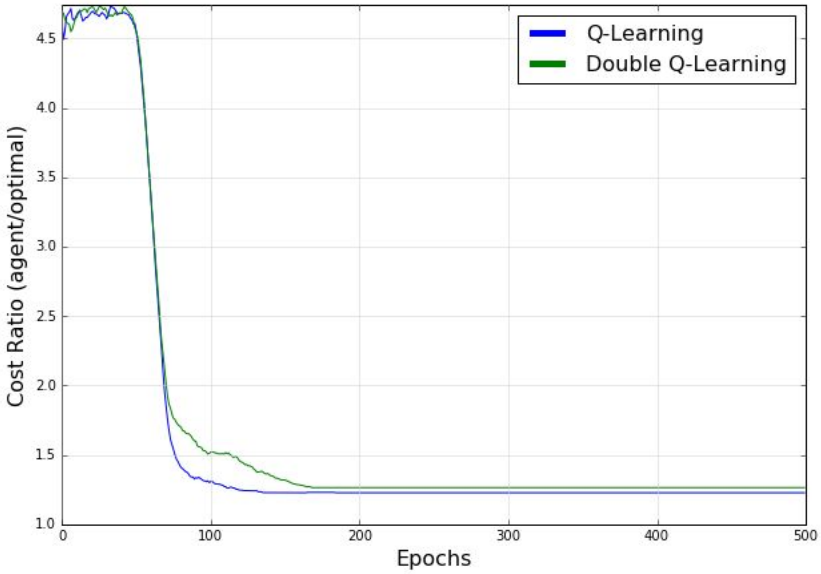
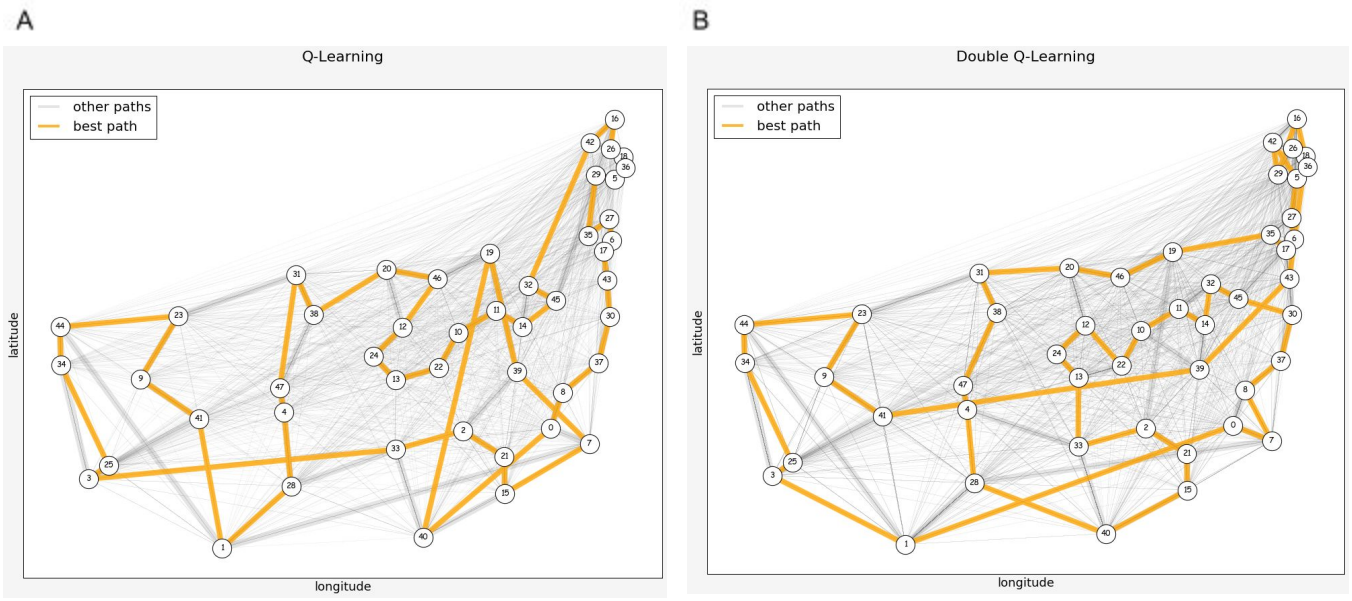


Figure 13. *Experimental results comparing learning performance between Q-learning and Double Q-Learning. The performance is measured as cost ratio (compared with optimal tour). Learning parameters for both learning methods were: epochs = 500, alpha = 0.9, gamma = 0.01, epsilon = 1, epsilon_decay = 0.001.*



C

Optimal tour compared with Q-Learning and Double Q-Learning best tours

start(0), 7, 37, 30, 43, 17, 6, 27, 5, 36, 18, 26, 16, 42, 29, 35, 45, 32, 19, 46, 20, 31, 38, 47, 4, 41, 23, 9, 44, 34, 3, 25, 1, 28, 33, 40, 15, 21, 2, 22, 13, 24, 12, 10, 11, 14, 39, 8, end(0)

start(0), 8, 37, 30, 43, 17, 35, 27, 5, 29, 26, 36, 18, 42, 16, 6, 45, 32, 11, 14, 39, 10, 22, 13, 24, 12, 20, 46, 19, 2, 21, 15, 40, 33, 4, 47, 38, 31, 23, 9, 41, 25, 3, 34, 44, 1, 28, 7, end(0)

start(0), 7, 8, 37, 30, 45, 32, 14, 11, 10, 22, 12, 24, 13, 33, 2, 21, 15, 40, 28, 4, 47, 38, 31, 20, 46, 19, 35, 17, 6, 27, 5, 42, 29, 16, 36, 26, 18, 43, 39, 41, 9, 23, 44, 34, 25, 3, 1, end(0)

Figure 14. *Experimental results comparing Q-learning and Double Q-Learning. A and B show the tour map for Q-Learning and Double Q-Learning respectively. The minimum cost tour learned by the agent is represented in orange, all other paths taken during training are in grey. C is the description of the minimum cost tour learned by the agent compared with the optimal tour. D shows the cost ratio performance of Q-Learning and Double Q-Learning.*

3. Analysis of results

Q-Learning has proven to be a proficient method for finding an optimal cost solution to the TSP in a simple environment with a small number of nodes. Initial results using a largely arbitrary set of default learning parameters allowed the agent to learn a near optimal route in under 5000 epochs. Experimenting with different α, γ, and λ values demonstrated that each of these parameters had a significant influence on the results. In terms of performance, the effect of different learning rates and discount factors was mostly limited to impacting the cost of the final tour route learned, with little impact on the speed at which the agent learning converged. Sections 2.1 and 2.2 demonstrated that, with the default parameters, smaller values of both α and γ tended to work best. Section 2.3 demonstrated that varying γ had a far more dramatic effect, impacting both epochs to convergence, and tour cost at convergence. Using a large decay rate of (λ =0.30) greatly increased speed to convergence, albeit without achieving an optimal solution. Section 2.4 demonstrated that with further tweaking of the α and γ parameters, and retaining this decay rate, an optimal cost tour route could be learned by the agent in just 109 epochs.

The experiments reported in sections 2.1 to 2.4 demonstrated that in such a simple environment, the effectiveness of Q-Learning in achieving a low-cost solution is largely insensitive to the parameter values chosen. While each parameter clearly affected the tour cost learned at convergence, the absolute range of this effect was small; with the exception of some extreme parameter values, virtually all parameterisations resulted in a near optimal solution given enough learning epochs.

Double Q-Learning fails to improve the learning performance compared with Q-Learning (fig.13). This results suggest that the maximization bias might not have a negative effect on this particular learning task or environment. Interestingly, the performance during Double Q-learning was particularly sensitive to the discount factor (γ). The value of γ determines the weight of the estimation used for the update of the Q matrix at every time step. Decreasing the values of γ dramatically improved the performance of Double Q-Learning.

Despite the good performance of the agent in the toy example, Q-Learning was not able to reach an optimal solution in a more complex environment. In the USA example, the agent showed a considerable improvement compared to a random walk. However, the learning performance converged to a suboptimal tour substantially different from the optimal tour.

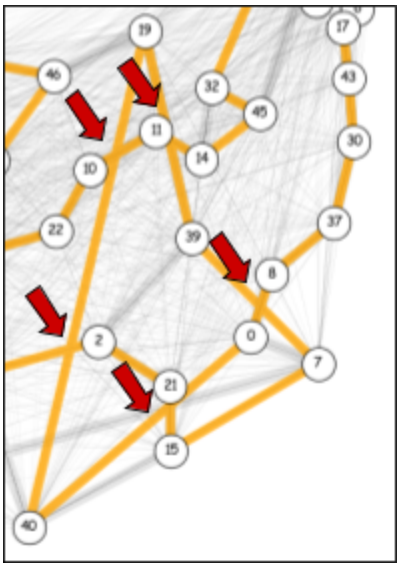


Figure 15. *Extract from USA environment tour map, showing the agent's best tour. The red arrows indicate the presence of crossovers of city-to-city paths.*

A visual analysis of the tours performed by the agent in the USA environment reveal relevant difference between the expected behaviour (optimal tour) and observed behaviour. The optimal tour contains no crossovers of city-to-city paths. Meanwhile, the best tour learned by the agent contains several of them (fig.15). This behaviour of the agent could be prevented by adding a heuristic rule into the learning algorithm.

In addition, the agent's tour shows longer paths between cities than the optimal tour. Hence the addition of a strong penalty for long paths between cities could, in theory, help the agent approximate the optimal performance.

4. Future work

The use of n-step Temporal Difference (TD) for the Q-matrix update might increase the chance the agent will learn an optimal tour. So far, all experiments were done using 1-step TD, namely the Q-matrix was updated at every time step. The value used for the update was calculated adding the immediate reward and the discounted estimation of the next state reward. However, this method has failed to return an optimum solution. Similarly, a Monte Carlo method where the Q-matrix is only updated at the end of every learning episode (epoch) is also likely to fail at reaching an optimum.

N-step TD (for 1 < n < No. cities) will also consider any intermediate method in which an Q-matrix update is done using ***n*** number of rewards following a given a state and time step (Sutton, R. S., 1988). For example, in a 2-step TD, the Q-matrix update value is calculated with the current reward, the discounted reward at the next time step and the discounted estimation of 2 time steps ahead.

N-step TD (for n=2,4,6,8,10) will be implemented in the U.S.A. scenario and the learning results compared with our best 1-step model (cost≅40000).

5. References

Applegate, D., Bixby, R., Chvatál, V., & Cook, W. (2006). The Traveling Salesman Problem: A Computational Study. Princeton University Press.

Hasselt, H. Van. (2010). Double Q-learning. Nips, pp.1–9.

Hasselt, H. Van, Guez, A. & Silver, D. (2015). Deep Reinforcement Learning with Double Q-learning. arXiv:1509.06461 [cs], (2), pp.1–5.

Sutton, R. S. (1988). Learning to predict by the method of temporal differences. Machine Learning, 3:9–44.

Sutton, R.S. & Barto, A.G. (2012). Reinforcement learning. Learning, 3(9), p.322.