University of Bielefeld

Faculty of Technology

Neuroinformatics Group

Report

Reinforcement Learning

Akhil Jain

Supervisors Prof. Dr. Helge Ritter
M.Sc. Sascha Fleer

September 26, 2017

Contents

1 Introduction

Reinforcement Learning works in a cycle of sense-action-goals i.e, how to map situations to actions and actions to rewards, so as to maximize the final reward of the goal state. Learners discover the action that yields most reward by exploring the possible actions. Actions may affect not only the immediate reward but also all the subsequent rewards. The two most important distinguishing features of Reinforcement Learning are first trial and error search and second delayed reward.

The importance of Reinforcement Learning in our lives today cannot be overstated. They are used virtually everywhere, from financial sector to manufacturing. Still the scope for innovation is truly vast. With the advancement in computing, this class of algorithms are certainly paving the way for the next artificial intelligence revolution.

1.1 Structure

The purpose of this project is to familiarize with the concept of Reinforcement Learning. Two algorithm strategies namely Z-Learning and Q-Learning are discussed in detail. Emphasis is on how these algorithms are effective in solving markov decision problems. In Q-Learning, discrete and unstructured markov decision problem is solved which has discrete state space and discrete control space. This discrete optimization problem is transformed into continuous optimization problem in Z-Learning, thus leading to the alternate formulation of Markov Decision Problem called as Linear Markov Decision problem (abbreviated as LMDP). Implementation of both the algorithms is demonstrated on the gridworld environment. In the end we summarize our results and compare these two strategies.

2 Theoretical Part

Our objectives in this section is to describe the Reinforcement learning problem in a broad sense and the key elements of the problem's mathematical structure, such as value functions and Bellman equations. Focus will be on how LMDP where minimization over the continuous control space is convex and analytically tractable can be solved i.e, How Bellman equation can be exactly transformed into a linear equation. [Torodov 2006]

2.1 Basics

2.1.1 Markov decision process

A Markov decision process (abbreviated as MDP) is a discrete-time state transition system. The environment of Reinforcement learning is described using MDP. Framework of MDP is given as :

- S, a set of states of the environment (where s ϵ S)
- A, a set of actions (where a ϵ A)
- P : S × S × A $\rightarrow [0,1]$, which specifies the passive dynamics. Transition from one state to another under the influence of the action possible in that state and is written as $P(s^{'}|s,a)$, where

$$\forall s \in S, \forall a \in A \sum_{\forall s^{'} \in S} P\left(s^{'}|s,a\right) = 1 \tag{2.1}$$

- R : S × A × S $\rightarrow$ R, where$R(s,a,s^{'})$ gives the expected reward when transitioning from state s to $s^{'}$

The transition probabilities of a state depends only the current state and not on the history of predecessor states, i.e., all the states should have the Markov Property.

Furthermore, the learner and decisionmaker is called an agent. Similarly, the thing it interacts with, comprising everything outside the agent, is called the environment which in our case is the Gridworld. [Sutton and Barto]

The goal of the agent is to maximize the cumulative reward. Selection of optimal action in each particular state lets us achieve our objective. For finding the best action, we develop a strategy (also known as "Policy") Π_t where t is the time at which the agent has policy Π. The agent improves the policy after each action taken.

2.1.2 Value function

If the sequence of rewards received after time step t is denoted as $R_{t+1}, R_{t+2}, R_{t+3}..$, the agent then seeks to maximize the expected return (or minimize if we take the cost when transitioning from one state to another), where the return R_t is defined as a specific function of the reward sequence. In the simplest case the return is the sum of the rewards:

$$R_t = R_{t+1} + R_{t+2} + R_{t+3} + .. + R_T \tag{2.2}$$

Discounting is an additional concept that needs to be discussed. According to this approach, the agent tries to select actions so that the sum of the discounted rewards it receives over the future is maximized.

$$R_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + ... = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \tag{2.3}$$

Discount rate is bounded to be smaller than 1, a mathematical trick to make an infinite sum finite and further helps in proving the convergence. If $\gamma \ll 1$ then agent considers the immediate reward and if $\gamma \approx 1$ then agent considers the all the rewards it receives over the future.

State-Value function estimates the importance of the agent to be in a given state. Its the expected reward when the agent is in the state s and strategy Π is pursued. This is denoted by $v_\Pi(s)$

$$v_\Pi(s) = E_\Pi[R_t | S_t = s] = E_\Pi[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} | S_t = s] \tag{2.4}$$

where $E_\Pi[.]$ denotes the expected value of a random variable given that the agent follows policy Π, and t is any time step.

2.1.3 Bellman equation

For any policy Π and any state s, the following consistency condition holds between the value of s and the value of its possible successor states:

$$\begin{aligned}
v_{\Pi} &= E_{\Pi}[R_t \,|\, S_t = s] \\
&= E_{\Pi}[R_t + \gamma R_{t+1} \,|\, S_t = s] \\
&= \sum_a \Pi(a|s) \sum_{s'} \sum_r p(s', r|s, a)[r + \gamma E_{\pi}[R_{t+1} \,|\, S_{t+1} = s']] \\
&= \sum_a \Pi(a|s) \sum_{s', r} p(s', r|s, a)[r + \gamma v_{\Pi}(s')] \qquad \forall s \in S
\end{aligned} \tag{2.5}$$

It expresses a relationship between the value of a state and the values of its successor states.

Principle of Optimality: An optimal policy has the property that whatever the initial state and initial decision are, the remaining decisions must constitute an optimal policy with regard to the state resulting from the first decision.[Bellman]

The Bellman optimality equation for v_{π}^{*} represents the value of a state s under an optimal policy equal to the expected return of the best action in this state.

$$v_{\Pi}^{*} = \min_{a \epsilon A_s} (\sum_{s', r} p(s', r|s, a)[r + \gamma v_{\Pi}(s')]]) \qquad \forall s \in S \tag{2.6}$$

2.1.4 Value iteration

The key idea of Dynamic Programming, and of Reinforcement Learning generally, is the use of value functions to organize and structure the search for good policies. We can find optimal policy once we find the optimal value function (2.5) Note that value iteration is obtained simply by turning the Bellman optimality equation into an update rule. The update rule after each iteration converges towards finding the optimal state value.

Note in the following algorithms, we considered cost instead of reward, thus we minimize instead of maximizing the update rule.

Algorithm 2: Value iteration

```
Initialize $V(s) = 0$.
while not converged do
    for all $s \in S$ do
        get all possible action in state s
        Update value function:
        $v^*_\Pi = \min_{a \epsilon A_s}(\sum_{s',r} p(s',r|s,a)[r + \gamma v_\Pi(s')])$
        Note that the KL divergence value is considered in calculating the reward
          such that the optimal value function equals to that calculated in Linear
          MDP case ($R(s) = R(s) + \lambda KL(a^*(.|s)||P(.|s))$ )
    end
end
```

2.2 Q Learning

Q Learning is a model free Reinforcement Learning technique. The parameters of the environment are generally unknown to us. The optimal action-value function is the expected return for using action a in a certain state s and is approximated by interacting with the environment. Typically, Monte Carlo or Temporal Difference Learning are used to solve this problem. Monte Carlo explores each action in every state and compares the reward in the goal state. The one with the highest reward (or minimum cost) will be the optimal action-state value. After large number of iteration, the agent can approximate the action-value function. In temporal difference whole set of actions are not considered, value function is re-evaluated after each action is taken. It forms a union of Dynamic Programming and Monte Carlo methods by learning from individual steps online. The core of Q learning is a simple value iteration update. In this scenario, we use One-step Q learning that has the update rule which is based on Temporal Difference Learning update rule. We can also use Q function to define the policy, because an action can be chosen just by taking the one with the maximum Q value for the current state.

The Q-learning rule is

$$Q(s_t, a_t) = Q(s_t, a_t) + \alpha_t[r_{t+1}(s, a) + \gamma \min_{a \epsilon A_{s_{t+1}}} Q(s_{t+1}, a) - Q(s_t, a_t)] \tag{2.7}$$

where α_t = learning rate
$r_{t+1}(s, a)$ = Immediate reward

$Q(s_{t+1}, a)$ = Q value for next state after action a

An essential part of Q-learning in finding an optimal policy is the selection of action a in state s. In Q-learning there exists a tradeoff between selecting the currently expected optimal action, or selecting a different action in the hope that it will yield a higher cumulative reward in the future. To investigate the goal-finding abilities of an agent using Q-learning, we investigate ϵ-greedy and random exploration strategies.

2.2.1 Random Q learning

This exploration strategy is the slowest among all other strategies. We randomly select the action and generally the agent takes a long time to converge. Selection of action can be done by generating a random number in between the total number of possible actions in that state. If the maze or the gridworld is large, to reach to the goal state, it takes a long time but this strategy is guaranteed to converge.[Torodov 2009]

Algorithm 3: Random Q learning

```
Initialize $Q(s,a) = 0$
for each episode do
    Move the agent to starting state
    while not equal to total no. of steps possible in each episode or agent reached the goal state do
        Get all the action possible on the state
        Get the next state by taking the random action
        $\alpha = \frac{c\alpha}{c\alpha + t}$
        Update Q value:
        $Q(s_t, a_t) = (1-\alpha)\, Q(s_t, a_t) + \alpha_t[r_{t+1}(s,a) + \gamma \min_{a \epsilon A_{s_{t+1}}} Q(s_{t+1}, a) - Q(s_t, a_t)]$
        where $\gamma$ is discount rate
        Move agent to the next state
        t=t+1
    end
    Increase the episode by 1
end
```

2.2.2 Greedy Q learning

ϵ-greedy exploration is one of the most used exploration strategies. It uses $0 \leqslant \epsilon \leqslant 1$ as parameter of exploration to decide performing action using Qt(st, a). The agent chooses the action with the highest Q-value in the current state with probability $1 - \epsilon$, or a random action otherwise. A larger value for ϵ means more exploration actions are selected by the agent. Randomness is necessary for an agent navigating through all the possible policies to learn the optimal policy. With the experiments, we will find out which value of ϵ is optimal for our setup and how ϵ-greedy performs compare to the random strategy.

Algorithm 4: Greedy Q learning

```
Initialize $Q(s,a) = 0$
for each episode do
    Move the agent to starting state
    while not equal to total no. of steps possible in each episode or agent reached the
     goal state do
        Generate a random number between 0 and 1
        if random number $\leqslant \epsilon$ then
            find out the minimum q value of next states possible
            Based on the state found, find out the action
        end
        else
            select the random action possible on the current state
        end
        Get the next state by taking the action
        $\alpha = \frac{c\alpha}{c\alpha + t}$
        Update Q value:
        $Q(s_t, a_t) = (1-\alpha)\ \text{Q}(\text{s}_t, a_t) + \alpha_t[r_{t+1}(s,a) + \gamma \min_{a \epsilon A_{s_{t+1}}} Q(s_{t+1}, a) - Q(s_t, a_t)]$
        where $\gamma$ is discount rate
        Move agent to the next state
        t=t+1
    end
    Increase the episode by 1
end
```

2.3 Z Learning

Transforming the discrete optimization problem over actions to a continuous optimization problem over transition probabilities is the main idea of Z learning. The orignal formulation of MDP had predefined uncontrolled transition probabilty which controlled the dynamics but Torodov came up with an alternate formulation of the MDP i.e, to view the controlled probability distribution as a stochastic policy over deterministic actions. This calculates the controlled probability based on the effect of action on passive dynamics of the environment. Framework of MDP in Z learning case:

- S, a set of states of the environment
- A_s is a set of admissible actions at state s ϵ S
- r(s,a) $\geqslant$ 0 is a reward for being in state s and choosing action a ϵA_s and q(s) is the immediate reward of state s
- $\bar{P}(a)$ is a stochastic matrix whose element $\bar{p}_{sj}(a)$ is the transition probability from state s to state j under action a

Next we define the controlled transition probabilities as

$$p_{sj}(a) = \bar{p}_{sj} exp(a_j) \tag{2.8}$$

This guarantees positive probabilities and absolute continuity of the controlled dynamics with respect to the passive dynamics. We initialize our stochastic controlled transition matrix with passive dynamics of the model, i.e, P(0) $=\bar{P}$. P(a) has two constraint, $p_{sj} = 0$ where $\bar{p_{sj}} = 0$ and sum of all the probabilities from one state equals 1 ($\sum_j \bar{p}_{sj} exp(a_j) = 1$). Since action effects directly on transition probabilty, the action cost term r(s,a) can be directly calculated by measuring the deviation between controlled and transition probabilty and adding it to the immediate reward.

$$r(s,a) = q(s) + [KL(p_s(a)||p_s(0))] = q(s) + \sum_{j:\bar{p}_{sj} \neq 0} p_{sj}(a) \log \frac{p_{sj}(a)}{p_{sj}(0)} \tag{2.9}$$

Substituting this equation with optimal value function 2.5, we get

$$v^*(s) = \min_{a \epsilon A_s} \{q(s) + \sum_j \bar{p}_{sj} exp(a_j)(a_j + v(j))\} \tag{2.10}$$

Minimization in this equation subject to constraint and is solved using Lagrange multipliers [Torodov 2006] which gives the optimal control law as

$$a_j^*(s) = -v(j) - \log(\sum_k \bar{p}_{sk} exp(-v(k))) \tag{2.11}$$

So, if we have the optimal value function we can easily calculate the optimal control law from the equation above. Thus finding the optimal control law indicates the next best state the agent should go to. Optimally controlled transition probability is given as

$$p_{sj}(a^*(s))) = \frac{\bar{p}_{sj} exp(-v(j))}{\sum_k \bar{p}_{sk} exp(-v(k))} \tag{2.12}$$

Substituting the optimal control in equation (2.10) and dropping the min operator, we get

$$\begin{aligned} v(s) =& q(s) + \sum_j \bar{p}_{sj}(a^*(s))(a_j^* + v(j)) \\ &= q(s) - \log(\sum_k \bar{p}_{sk} exp(-v(k)) \end{aligned} \tag{2.13}$$

Taking exponential on both sides and rearranging the terms we get

$$\begin{aligned} z(s) &= exp(-q(s)) \sum_j \bar{p}_{sj} z(j) \\ where \quad z(s) &= exp(-v(s)) \end{aligned} \tag{2.14}$$

2.3.1 Z iteration

In equation 2.14, the Z-learning algorithm is presented as a way to solve the linear eigen vector problem $z = G\bar{P}z$ where G is the diagonal matrix with $\exp(-q(s))$ as its diagonal element, and $\sum_j \bar{p}_{sj}$ is nothing but passive dynamics. One way to solve this equation is by iterating it until the z value converges. The reason it converges is because the stochastic matrix $\bar{P}$ has spectral radius 1 and multipication with G boils it down to a value $\leqslant 1$. Iteration is equivalent to the power method (without the rescaling) so it converges to a largest eigen vector with eigenvalue 1.

Algorithm 6: Z iteration

```
Initialize $Z(s) = 1$ and $G = \exp(-q(s))$ where s represent the state.
**while** *not converged* **do**
    **for** *all $s \in S$* **do**
        Update Z value: $Z_{t+1}(s) = G(s) * P(.|s) * Z_t(s)$
    **end**
**end**
```

2.3.2 Random Z Learning

Random Z-learning samples transitions from the passive dynamics P, which essentially amounts to a random walk. Since LMDPs have no explicit actions, each transition (s_t, r_t, s_{t+1}) consists of a state s_t, a next state s_{t+1} and a reward r_t recorded during the transition. Z-learning maintains an estimate Z of the optimal Z value, and this estimate is updated after each transition. From (2.14), the update rule can easily be derived as:

$$z(s_t) = (1 - \alpha)z(s_t) + \alpha \exp(r_t) z(s_{t+1}) \tag{2.15}$$

Algorithm 7: Random Z learning

```
Initialize $Z(s) = 1$
**for** *each episode* **do**
    Move the agent to starting state
    **while** *not equal to total no. of steps possible in each episode or agent reached the goal state* **do**
        Find out all the next possible states
        Randomly select the next state
        $\alpha = \frac{c\alpha}{c\alpha + t}$
        $z(s_t) = (1 - \alpha) * z(s_t) + \alpha * \exp(r_t) * z(s_{t+1})$
        Move agent to the next state
        t = t+1
    **end**
    Increase the episode by 1
**end**
```

2.3.3 Greedy Z Learning

In Z learning, the expected reward of the next state is dependent on the passive dynamics of the model. In greedy Z learning, we sample according to the optimal control probability (To sample according to more informed transition probability) (2.12) and thus the states with larger estimated Z-values are more frequent which speeds up the approximation and thus coverges faster. States with higher Z value will have a control probability which is higher than or equal to the passive transition probability. Note that the importance sampling is done using the control probabilty to guide exploration by sampling transitions from a more informed distribution.[Jonsson] Thus for estimating properties of the passive dynamics and to speed up the convergence, samples are generated from the control probability distribution. The adjustment factor or the likelihood ratio is given as:

$$\frac{\bar{p}_{sj}}{p_{sj}(a^*(s))} \tag{2.16}$$

Thus the update rule becomes

$$z(s_t) = (1-\alpha)z(s_t) + \alpha\exp(r_t)z(s_{t+1})\frac{\bar{p}_{sj}}{p_{sj}(a^*(s))} \tag{2.17}$$

Algorithm 8: Greedy Z learning

```
Initialize $Z(s) = 1$
**for** *each episode* **do**
    Move the agent to starting state
    **while** *not equal to total no. of steps possible in each episode or agent reached the goal state* **do**
        Calculate the control probability of the state (see 2.12)
        Find out next state by sampling the control probability
        $\alpha = \frac{c\alpha}{c\alpha + t}$
        z(s$_t$) $= (1-\alpha)z(s_t) + \alpha\exp(r_t)z(s_{t+1})\frac{\bar{p}_{sj}}{p_{sj}(a^*(s))}$
        Move agent to the next state
        t = t+1
    **end**
    Increase the episode by 1
**end**
```

Implementation 3

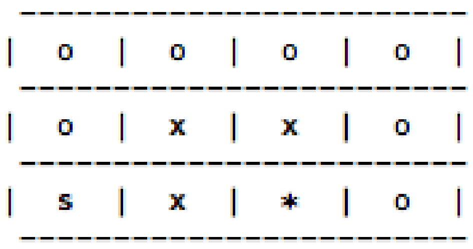
3.1 Overview

The purpose of the experiment is to compare the convergence rate of the algorithms and to find out which algorithm performs the best under different set of environment. We chose the environment of grid world which is often used as a toy environment for describing and understanding the reinforcement literature. In this experiment, the algorithms are evaluated on two different sizes of the grid so as to find out the performance when the complexity of the environment increases, i.e., size of the grid increases.

We start by evaluating the proposed algorithm on a finite state and action spaces. The state space is given by a N $\times$ N grid with some obstacles. The agent can move the state to any adjacent ones not occupied by an obstacle and the move succeeds with a transition probability. Additionally, a target state was defined and the agent incurs a cost of 1 at all states and obstacles has cost of 5 other than the target where it incurs no cost, i.e, 0.

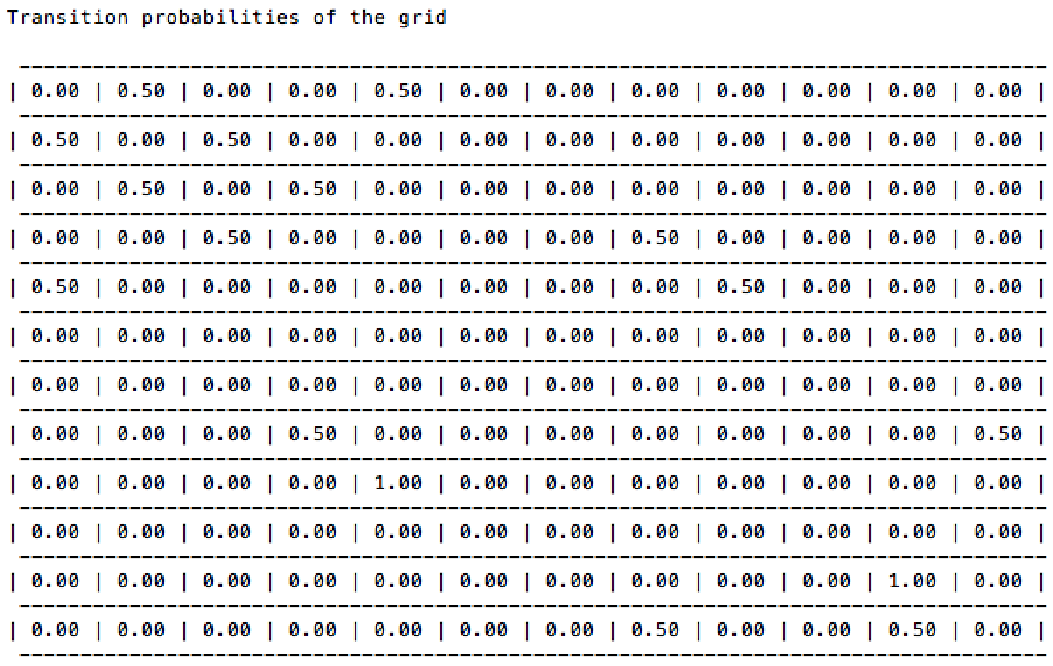
There are a number of different ways to organize the task of the agent. We use the episodic formulation. The agent's experience is grouped into episodes or trials. At the beginning of each trial, the environment is reset (agent moves back to starting state). The agent interacts with the environment until the end of the episode or until it reaches the goal state, then the environment is reset again and the cycle continues. The agent is aware of the episode resets. We are using fixed-length episodes and each episode has fixed number of steps the agent can take. A step is equivalent to moving from one state to another. Learning rate (α) is dynamic and decay at each step.

In the grid world the transition probabilty of each state has been calculated according to the number of states it can go to and each row indicates the probability of each state. As mentioned in the equation (2.1) it sums up to 1. For example, the state 1 can go right or go down. So the state agent can go to i.e, 2 and 5 has the probabilty of 0.5 and rest has 0 probability. The transition probability matrix for the above grid is shown below



```
-------------------------
|  o  |  o  |  o  |  o  |
-------------------------
|  o  |  x  |  x  |  o  |
-------------------------
|  s  |  x  |  *  |  o  |
-------------------------
```

Fig. 3.1: 3 × 4 grid state. The letter s indicates the starting state, x is the obstacle and * indicates the final or goal state of the grid



```
Transition probabilities of the grid

 -----------------------------------------------------------------------------------
| 0.00 | 0.50 | 0.00 | 0.00 | 0.50 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
 -----------------------------------------------------------------------------------
| 0.50 | 0.00 | 0.50 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
 -----------------------------------------------------------------------------------
| 0.00 | 0.50 | 0.00 | 0.50 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
 -----------------------------------------------------------------------------------
| 0.00 | 0.00 | 0.50 | 0.00 | 0.00 | 0.00 | 0.00 | 0.50 | 0.00 | 0.00 | 0.00 | 0.00 |
 -----------------------------------------------------------------------------------
| 0.50 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.50 | 0.00 | 0.00 | 0.00 |
 -----------------------------------------------------------------------------------
| 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
 -----------------------------------------------------------------------------------
| 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
 -----------------------------------------------------------------------------------
| 0.00 | 0.00 | 0.00 | 0.50 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.50 |
 -----------------------------------------------------------------------------------
| 0.00 | 0.00 | 0.00 | 0.00 | 1.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
 -----------------------------------------------------------------------------------
| 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
 -----------------------------------------------------------------------------------
| 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 1.00 | 0.00 |
 -----------------------------------------------------------------------------------
| 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.50 | 0.00 | 0.00 | 0.50 | 0.00 |
 -----------------------------------------------------------------------------------
```

Fig. 3.2: Transition probability matrix

Result 4

For comparison, we use two different size of grid. Accordingly, we have changed the parameters.

4.1 Small grid (3X4)

Approximation error for each step is calculated as:

$$\frac{\max_i |v(i) - \hat{v}(i)|}{\max_i \hat{v}(i)} \tag{4.1}$$

where $\hat{v}$ is the optimal value of state obtained from the value iteration and v is the value of the state at each step. [Torodov 2006] Heatmap for the graphical representation of the value of state was chosen. It uses colors to communicate the relationship between data and thus making it very easy for identifying the important state. A full range of hues exist between white and dark red in heatmap cells, white color has low state value and red has high state value. Black indicates the blocking state of the grid.

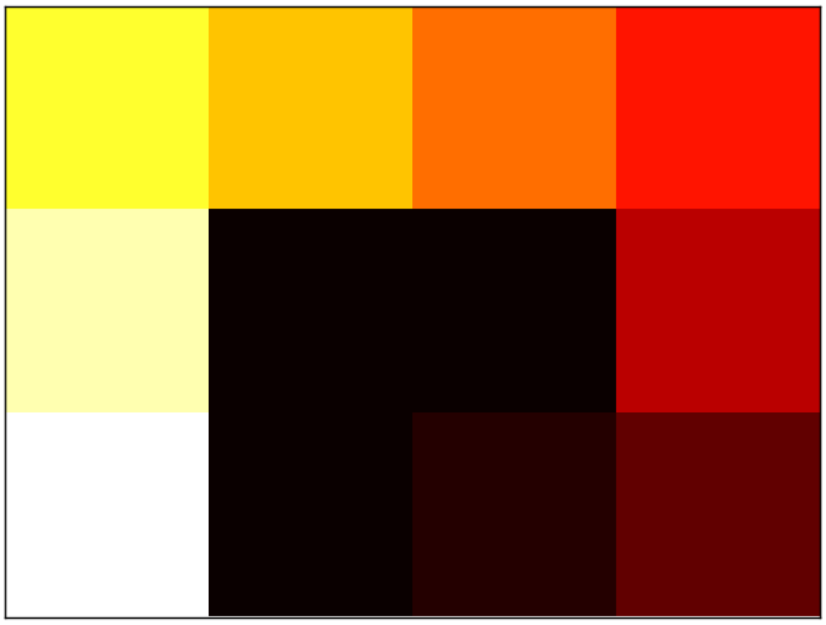


Fig. 4.1: Heatmap of a 3X4 grid obtained from value iteration.

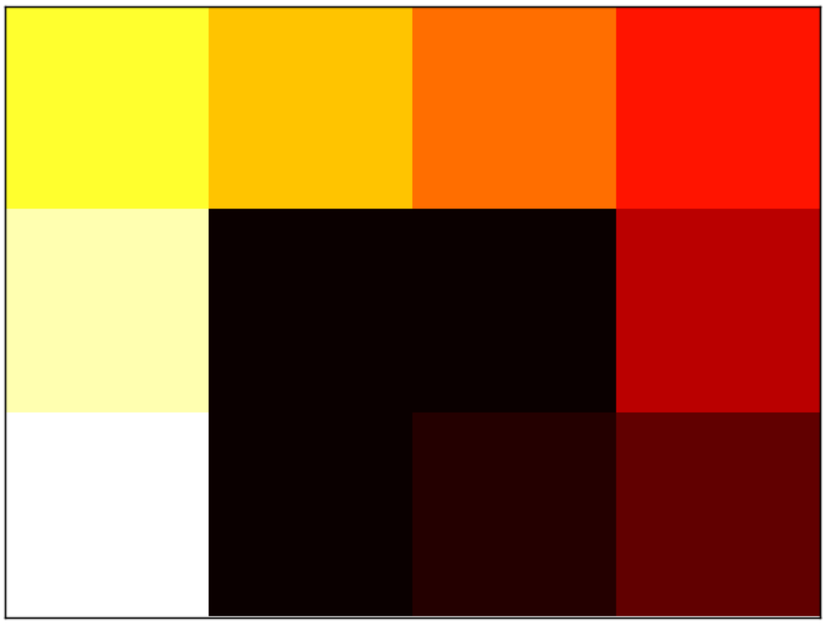


Fig. 4.2: Heatmap obtained from Z iteration

The total number of episodes were 10 and in each episode we had 100 steps. The constant $c\alpha$ was 100 for both the algorithms. Q learning requires exploration. Thus, we implement an ϵ-greedy policy with $\epsilon = 0.1$. Z learning implicitly contains exploration so we directly use the greedy policy. Entire simulation was repeated 5 times.

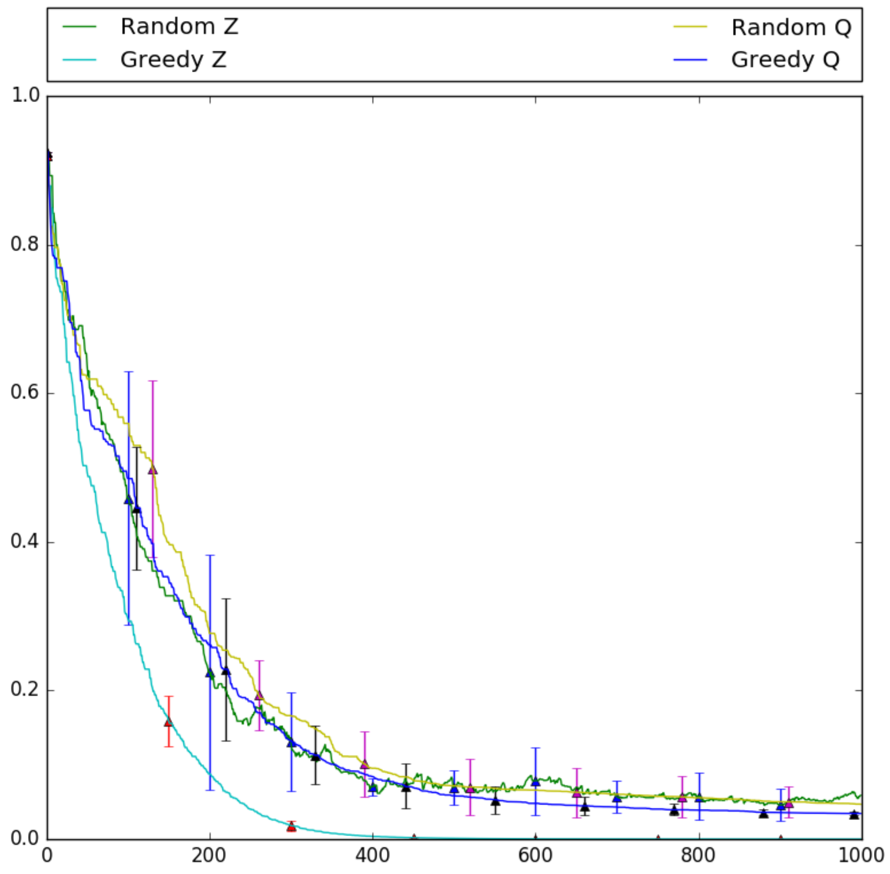


Fig. 4.3: Comparision in small grid case

Both algorithms converge quickly against the optimal state-value function. Importance sampling speeds up convergence slightly in the small Gridworld. Z learning performs slightly better but overall they all converge fastly, which can be contributed to the fact that the exploration in small grids are limited and thus agent in all the cases behave almost the same way, i.e, they follow more or less the same pattern.

4.2 Big grid (11X11)

Heatmap for the grid obtained from Z iteration is shown in Figure 4.4. The total number of episodes were 50 and in each episode we had 1000 steps. The constant

$c\alpha$ was optimized separately for each algorithm. Its value for calulating the learning rate was 5000 for Z learning and 10000 for Q learning. We use an ϵ-greedy policy with $\epsilon = 0.2$. Finding the optimal parameters is of critical importance. Trying out different values for parameter may give some idea about it but it takes a long time. For example, complexity of the grid in our case determined the learning rate, the higher the complexity the higher the learning rate. Learning rate plays a very crucial role in deciding the convergence and thus should be chosen wisely. The entire simulation was repeated 5 times and the difference in the behaviour of the agent each time was plotted by the error bar.

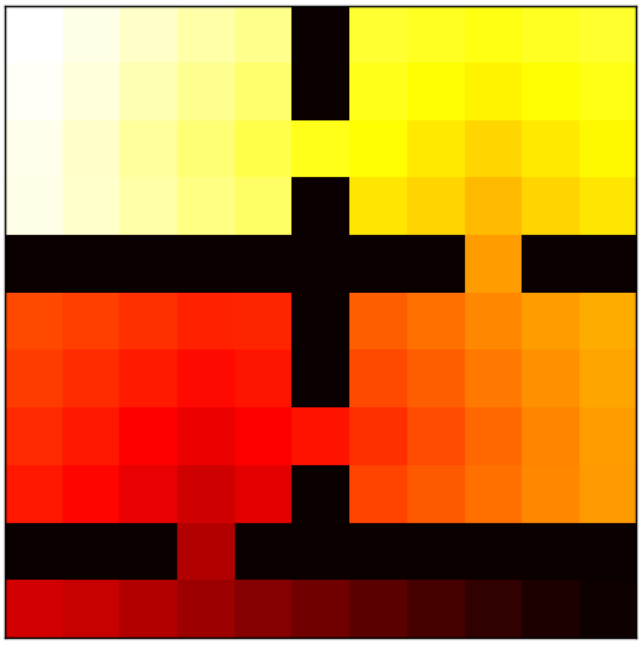


Fig. 4.4: Heatmap of 11X11 grid (generated from the Z iteration obtained state values)

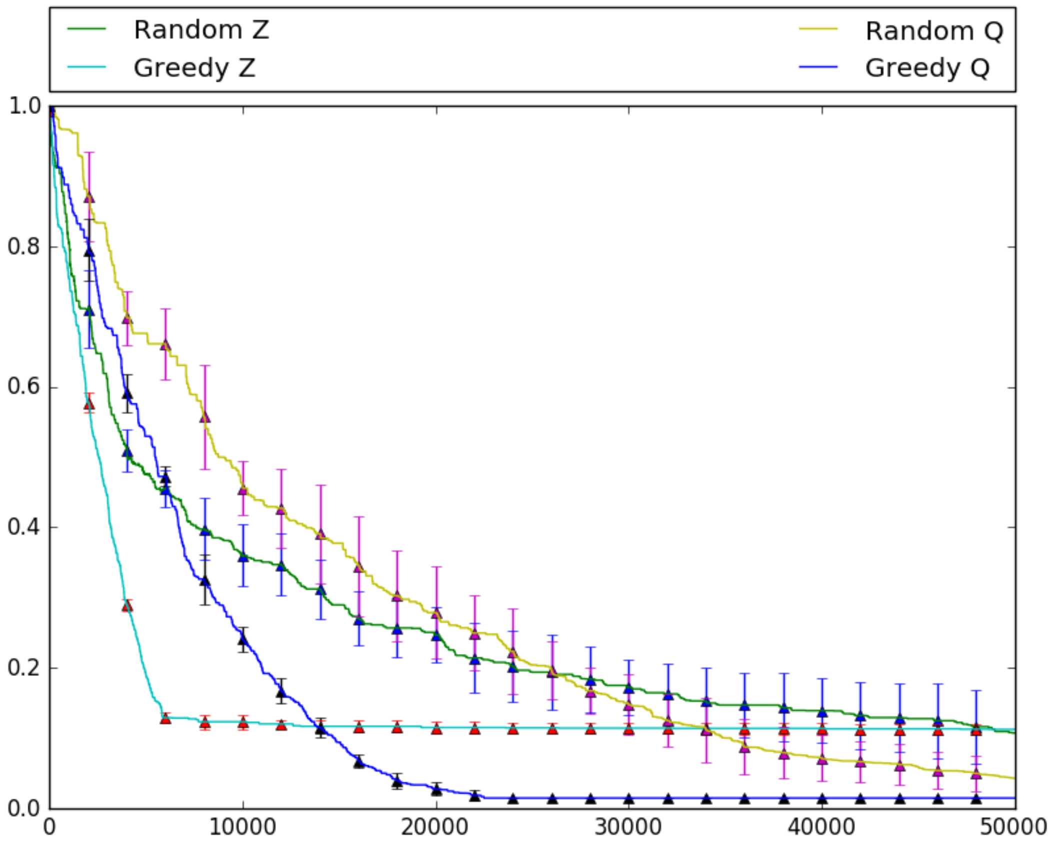
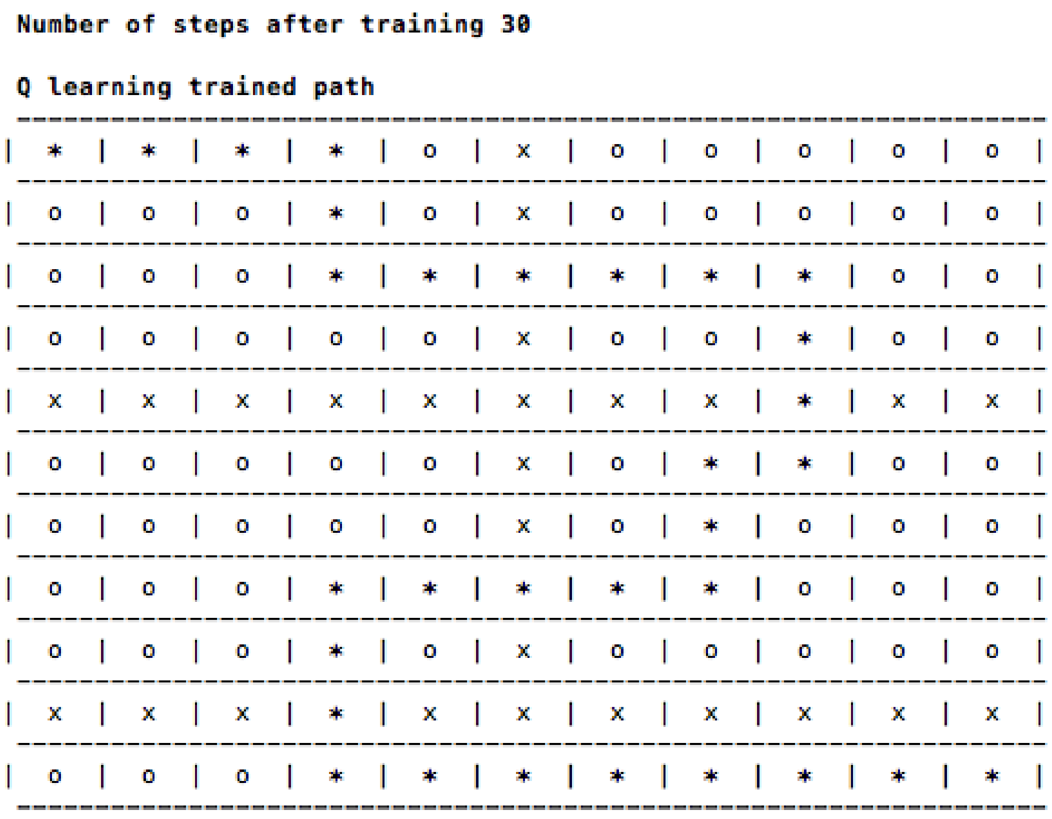


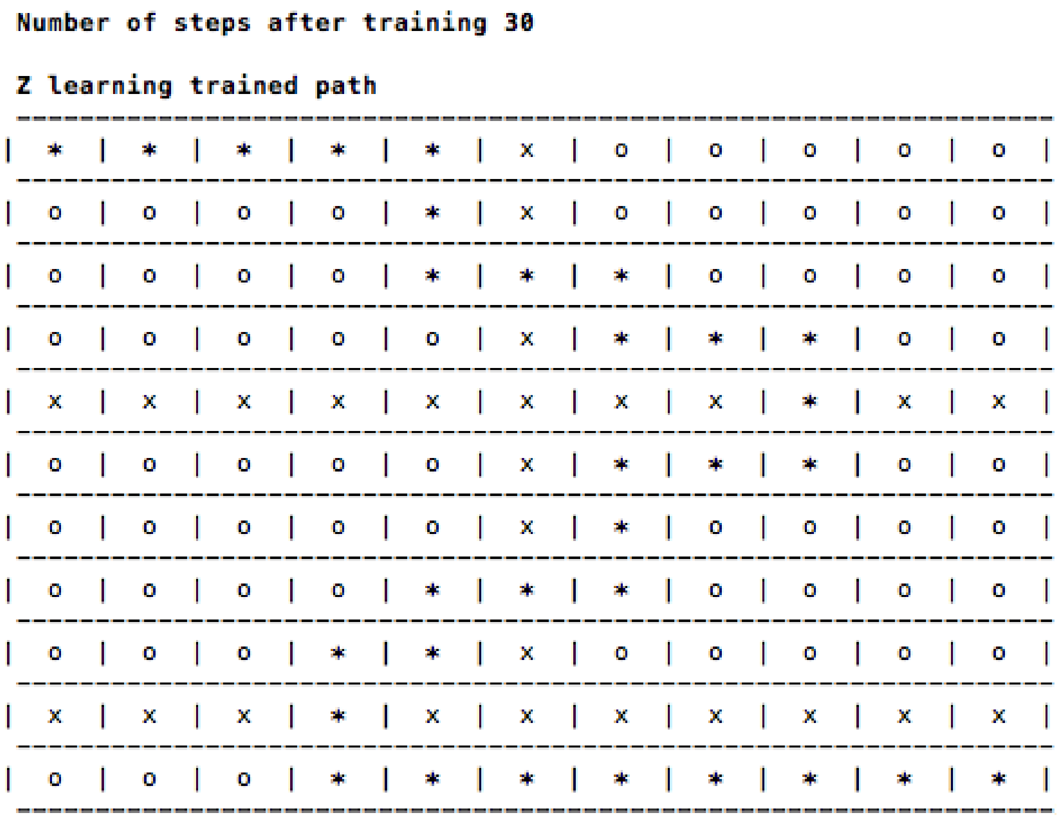
Fig. 4.5: Comparision in big grid case



```
Number of steps after training 30

Q learning trained path
-----------------------------------------------------------------
| * | * | * | * | o | x | o | o | o | o | o |
-----------------------------------------------------------------
| o | o | o | * | o | x | o | o | o | o | o |
-----------------------------------------------------------------
| o | o | o | * | * | * | * | * | * | o | o |
-----------------------------------------------------------------
| o | o | o | o | o | x | o | o | * | o | o |
-----------------------------------------------------------------
| x | x | x | x | x | x | x | x | * | x | x |
-----------------------------------------------------------------
| o | o | o | o | o | x | o | * | * | o | o |
-----------------------------------------------------------------
| o | o | o | o | o | x | o | * | o | o | o |
-----------------------------------------------------------------
| o | o | o | * | * | * | * | * | o | o | o |
-----------------------------------------------------------------
| o | o | o | * | o | x | o | o | o | o | o |
-----------------------------------------------------------------
| x | x | x | * | x | x | x | x | x | x | x |
-----------------------------------------------------------------
| o | o | o | * | * | * | * | * | * | * | * |
-----------------------------------------------------------------
```

Fig. 4.6: Q learning optimal path



```
Number of steps after training 30

Z learning trained path
-----------------------------------------------------------------
| * | * | * | * | * | x | o | o | o | o | o |
-----------------------------------------------------------------
| o | o | o | o | * | x | o | o | o | o | o |
-----------------------------------------------------------------
| o | o | o | o | * | * | * | o | o | o | o |
-----------------------------------------------------------------
| o | o | o | o | o | x | * | * | * | o | o |
-----------------------------------------------------------------
| x | x | x | x | x | x | x | x | * | x | x |
-----------------------------------------------------------------
| o | o | o | o | o | x | * | * | * | o | o |
-----------------------------------------------------------------
| o | o | o | o | o | x | * | o | o | o | o |
-----------------------------------------------------------------
| o | o | o | o | * | * | * | o | o | o | o |
-----------------------------------------------------------------
| o | o | o | * | * | x | o | o | o | o | o |
-----------------------------------------------------------------
| x | x | x | * | x | x | x | x | x | x | x |
-----------------------------------------------------------------
| o | o | o | * | * | * | * | * | * | * | * |
-----------------------------------------------------------------
```

Fig. 4.7: Z learning optimal path

In Figure 4.5, the speed difference becomes clearer. Greedy Z algorithm is the fastest and the Greedy variant is relatively faster than the random variant. Greedy Z learning doesnt quite converge in the big grid, because we sample according to the control probability and few of the unwanted states were not easily reachable in this grid. However, the agent reaches the goal in the minimum number of steps. The path learned by Q learning and Z learning are a bit different in this grid (Figure 4.6 and Figure 4.7) but both have the optimal path and reached the goal state in the same number of steps. Also in Figure 4.5, the error bar in random case is high which indicates the random behavour of the agent where as in greedy algorithms it follows the same path most of the time.

We also checked the algorithm on easy grid having the same size as above, lowering the complexity does not give any difference in the result which indicates the greedy nature of Z learning (Figure 4.9). As it is implicitly greedy, we can not converge it by adding some randomness as we did in the case of Q learning where we explore and exploit based on the ϵ value.

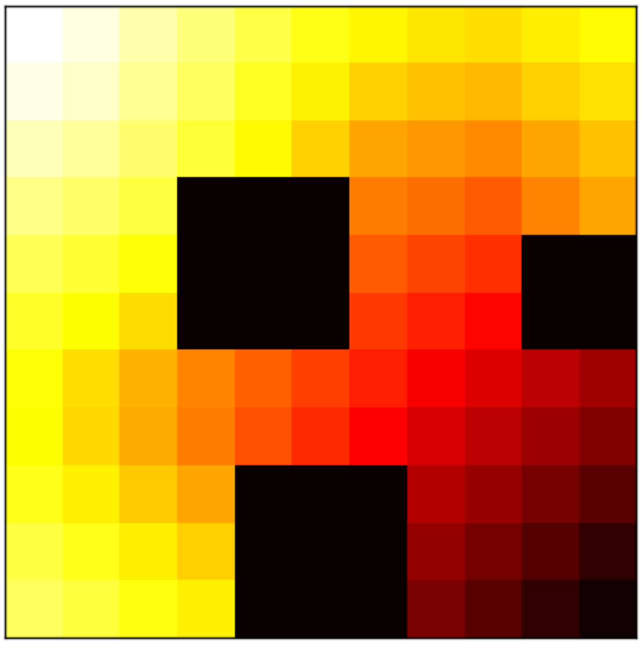


Fig. 4.8: Heatmap of 11X11 grid (Z iteration obtained state values)

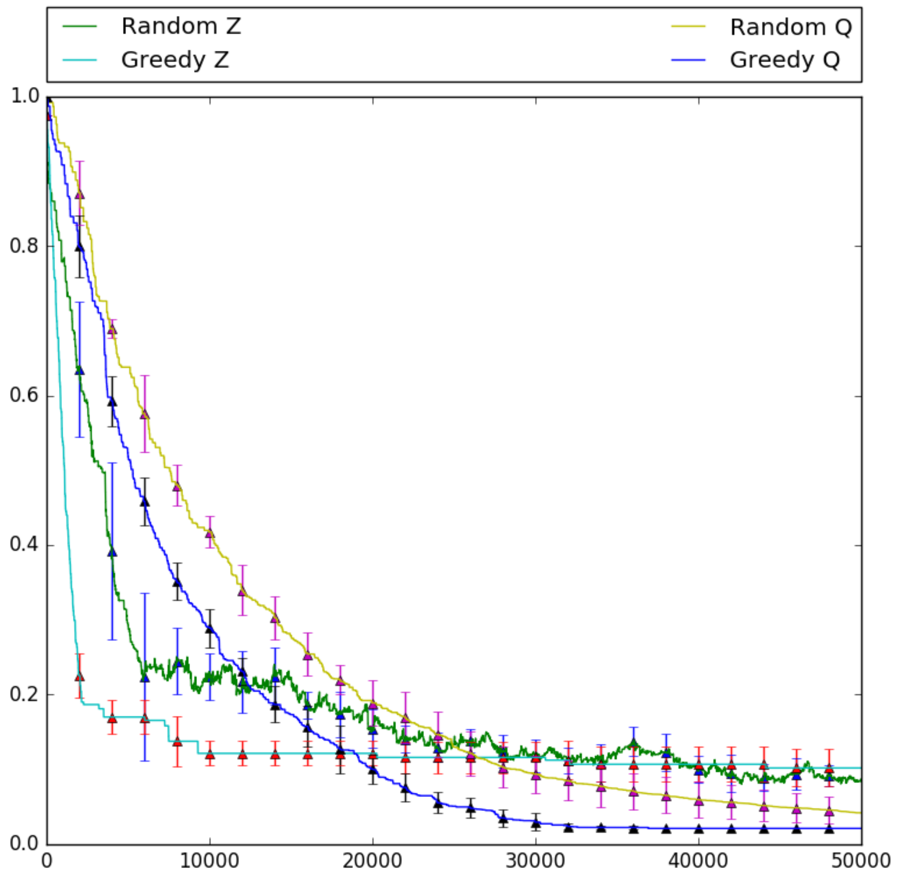


Fig. 4.9: Comparision in less complex grid

5 Conclusion

The task of finding out the shortest path in a grid can be solved using various approaches however it is not wise to settle for the first that comes to mind. Z Learning is the fastest approach, however the complete information about the environment or the passive dynamics of the model is required. Q Learning overcome the limitation of Z Learning through random exploration but this exploration comes with the penalty of the slow convergence.

5.1 Future Work

Further inquiry could be based on moving from two dimensional world to three dimensional world. It would be really interesting to see how the algorithms behave when we further increase the complexity of the environment.

Bibliography

[Torodov 2009]	*Efficient computation of optimal actions*. Nov. 2009 (cit. on p. 7).
[Torodov 2006]	*Linearly-solvable Markov decision problems*. Mar. 2006 (cit. on pp. 3, 10, 15).
[Sutton and Barto]	Sutton Sutton and Barto Barto. *Reinforcement Learning: An Introduction*. The MIT Press, 2017, p. 1 (cit. on p. 4).
[Bossert]	*Z-Learning auf diskreten Markov-Entscheidungsproblemen mit zustandsabhangigen Aktionen*. Nov. 2013.
[Froese]	*Vergleich von Z-Learning und Q-Learning auf diskreten Markov Decision Problems*. Nov. 2010.
[Jonsson]	*Hierarchical Linearly-Solvable Markov Decision Problems*. Mar. 2016 (cit. on p. 12).
[Bellman]	*Bellman, R.E. 1957. Dynamic Programming*. Princeton University Press, Princeton, NJ. Republished 2003: Dover, ISBN 0-486-42809-5, Mar. 1957 (cit. on p. 5).