Robotic Arm Path Planning using Optimal Control

Siddhartha Cheruvu, Aaron Arul Maria John, Lei Zhang

***Abstract*—The increase in automation in modern industries resulted in increased utilization of industrial robotic arms. Robotic arms play a vital role in improving the efficiency of several industrial processes such as manufacturing, assembly etc. In order to achieve the given task, the process of developing an optimal path to maneuver the robotic arms is crucial. This is termed as *'Path Planning'*. An ideal path takes the robotic arm from an initial configuration (start state) to the desired final configuration (goal state) under given constraints in the task environment. Path planning algorithms typically generate a set of intermediate points between the start and end states to achieve the given task. In this report, path planning is treated as a sequential decision making problem (Markov Decision-making Problem or MDP). Methods such as *'Policy Iteration'*, *'Value Iteration'* and *'Q-Learning'* are implemented to develop the solution. Finally the results of these algorithms are compared against each other and the conclusions made are provided at the end of this report.**

***Index Terms*—Robotic arm, Path planning, MDP, Policy iteration, Value iteration, Q-Learning**

I. Introduction

The configuration of a robotic arm is generally described by the position and the orientation (pose) of its end-effector. Navigation of the end-effector from an initial configuration to a desired pose is critical in maneuvering the robotic arms. To achieve this, the changes in the pose are described as a continuous function of time which is termed as *'path'*. The path indicates the set of points between the start and end locations. It doesn't hold any notion of time. On the other hand, when the notion of time is introduced to a path in the form of velocity and accelerations at the way points it is called a *'trajectory'*. So, path planning can be treated as a kinematic approach whereas trajectory planning is a dynamic approach.

In this project, optimal path of robotic arms subject to constraints are identified using policy iteration, value iteration and q-learning techniques. MATLAB and Python are the tools primarily used for this purpose. 2-Link and 3-Link planar arms with revolute joints are mainly considered in this analysis. Fig. 1 shows a typical 3-link planar arm.

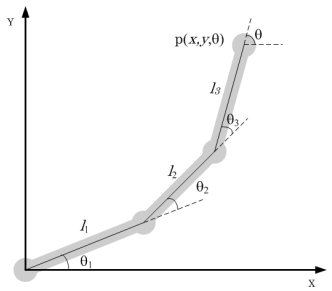

Fig. 1. 3-Link Planar Arm

II. Methods

The methods used for path planning in this project are

Offline Planning:

1) Value Iteration
2) Policy Iteration

Online Learning:

1) Q-Learning (Off-Policy Learning)

MATLAB is used to implement the above methods. Following sections of this report provide detailed explanation of the above methods used along with the modeling parameters and other hyper parameters. The convergence of different methods is also discussed. In each section, a brief explanation of the algorithm is provided, along with the implementation techniques and the discussion of results.

III. Implementation

A. Problem Formulation

In this section, the details of problem formulation for a 2-link planar arm are provided. 3-link arm uses a similar approach.

1) State Space: The arm moves only in the x-y plane. The configuration of the arm at any given time can be completely described by its two joint parameters θ_1 and θ_2. Hence, they are chosen to be the system states.

$$\mathbf{X} = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} \theta_1 \\ \theta_2 \end{bmatrix} \tag{1}$$

The joint angles are assumed to be unconstrained.

$$\theta_1 \in [-\pi, \pi],\ \theta_2 \in [-\pi, \pi]$$

Since path planning does not consider the dynamics of motion, the constraints for input torque are irrelevant in this problem formulation. Further, each state is quantized into 101 discrete values. Hence there are more than 10000 states in total. For a 3-link arm, the size of state space is greater than 1000000.

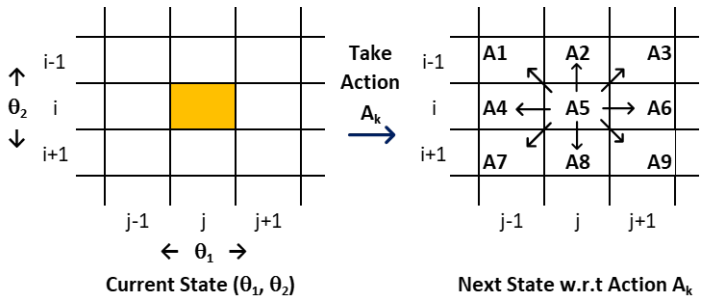

Fig. 2. Definition of Actions

2) Action Space: From any given state [θ_1, θ_2], 9 possible actions are identified for the arm. These 9 actions are defined based on the change that they would produce to the current state. Fig.2 provides the details of actions defined for a 2-link arm.

Each of these 9 actions can change the joint angles by one grid size of quantized states at maximum. For example, if the arm is at [$\theta_1(i)$, $\theta_2(j)$] at a given time, action A_1 would transform the states to [$\theta_1(i-1)$, $\theta_2(j-1)$] i.e., both the angles are decreased by one grid size.

Note: Action A_5 does not change the joint angles. It is analogous to 'Wait' action.

For a 3-link planar arm, the total number of actions possible from a given state is 27.

3) Transition Function: Actions are considered to be deterministic i.e., if an action is taken from a given state, then the resulting state can always be predicted with 100% certainty.

$$\mathbf{T} = 1, \quad \forall\ (s, a, s') \tag{2}$$

4) Reward Function: Goal state is given a positive reward (for example, +10). All the other non-terminal states are given a negative living reward or penalty (for example, -0.5). The negative reward at other states forces the algorithm to find the shortest path to the goal from the start state.

B. Method 1: Value Iteration

Value iteration is a sequential decision making approach that concerns with finding the optimal policy π using an iterative application of the Bellman's optimality equation. In each step, the value iteration algorithm synchronously updates the state value function for all the states, ie, update $V_{k+1}(s)$ from $V_k(s)$. The convergence criteria is set to the condition when the values of all states remain the same for two consecutive iterations. Value iteration is guaranteed to converge to an optimal policy for finite MDPs with a discounted reward. In value iteration, there is no explicit policy defined at the start of algorithm. Following equations present a basic layout of value iteration approach.

$$V_{k+1}(s) = \max_a (R(s,a) + \gamma \sum_{s'} V_k(s')) \tag{3}$$

$$V_0(s) = 0 \quad (\forall\ non-terminal states) \tag{4}$$

$$V_0(GoalState) = +10 \tag{5}$$

$$LivingReward,\ r = -0.2 \tag{6}$$

$$\gamma = 1 \tag{7}$$

C. Method 2: Policy Iteration

The process of policy iteration can be divided into two routines.

1) Policy Evaluation (Value Determination)
2) Policy Improvement

1) Policy Evaluation: In this phase, the current policy in each iteration is evaluated through an iterative approach which uses Bellman's equation and the corresponding state values are identified. The values are initiated as zeros. They are iterated until they converge for the current policy.

2) Policy Improvement: Based on the values evaluated in the policy evaluation phase, the policy improvement phase identifies the action that maximizes the value for each state. The next iteration's policy is chosen based on these max values at each state.

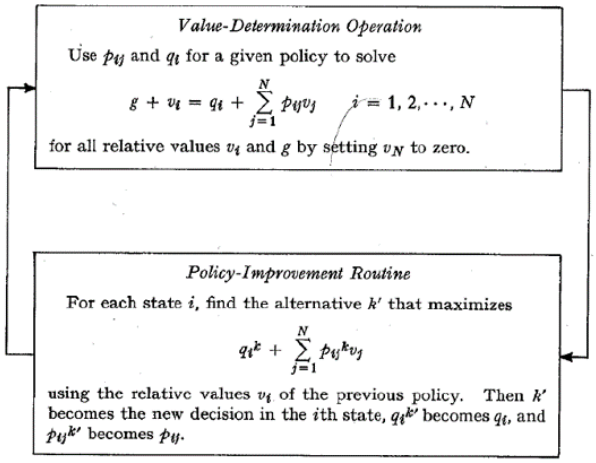

Fig. 3. Policy Iteration Cycle (Ref. [1])

Policy iteration obtains a sequence of continually improving policies and value functions, where we do policy evaluation and improvement separately. Every policy improvement is guaranteed to be an improvement over the current policy. The optimal policy is reached when the policies on two successive iterations are identical.

The parameters chosen for policy iteration are the same as those from the value iteration. The only modification is that there is an initial policy specified.

D. Method 3: Q-Learning

Q-Learning is an Online Learning approach. In contrast to offline planning, online learning methods rely on the experiences of the agent in each step to learn and solve the model. The agent needs to actually act and figure out the model based on the corresponding rewards. The state transition and reward function data is not generally available to the agent. The agent has to try out all the states and actions to learn the model.

Q-learning is an off-policy Temporal Difference algorithm developed in late 1980s. Following conditions generally lay the premise of a Q-learning problem.

1) The agent doesn't know the transition function T(s, a, s')
2) The agent doesn't know the reward function R(s, a, s')
3) The actions can be chosen by the learner to explore the grid world.
4) Goal is to learn the optimal policy and values.

The main idea behind Q-learning is that the agent learns from every experience. The action values Q(s,a) are updated

every time the agent experiences a transition (s,a,s',r). The outcome of each experience is termed as a sample. The Q-values are updated after each experience based on the following equation.

$$sample = R_{t+1} + \gamma \max_{a} Q(S_{t+1}, a) \tag{8}$$

$$Q(S_t, A_t) = (1 - \alpha)Q(S_t, A_t) + \alpha[sample] \tag{9}$$

The learning rate α is chosen as 1/episode so that the learning rate decays as the time progresses.

The learned action-value function, Q, directly approximates q*, the optimal action-value function, independent of the policy being followed. This dramatically simplifies the analysis of the algorithm and enabled early convergence proofs. The policy still has an effect in that it determines which state-action pairs are visited and updated. However, all that is required for correct convergence is that all pairs continue to be updated.

The exploration policy used in this project is ϵ-greedy approach. At every state, in order to choose an action, a random number is generated between 0 and 1. If the value of this random number is less than ϵ, then the action is chosen at random, otherwise, the action is based on the maximum Q-value. ϵ is a hyper-parameter that can be varied by the user. For this project, a decaying ϵ value is used.

IV. Simulation

A. Task Environment

The simulation of the robotic arm environment is done in MATLAB using Peter Corke's toolbox (Ref.[2]). An array of states (joint angles) is created based on the optimal policy from the start state. Each row of this array corresponds to the joint angles at a given time. The motion of arm can be simulated when this array is provided as an input to the robotics toolbox.

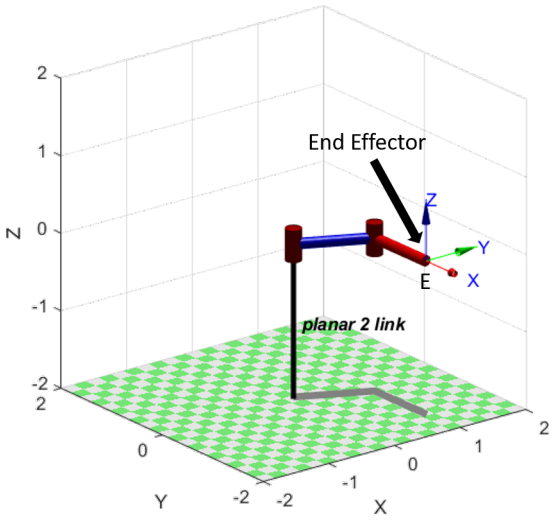

Fig. 4. Modeling Environment in MATLAB (Ref.[2])

B. Obstacle Modeling

In few of the iterations, obstacles are also modeled in the environment. They are simulated as planar n-dimensional polygons in the reachable workspace of the robotic arm. Then, on each link of the arm, several points are created using linear interpolation. A collision is described as a configuration where at least one of these link points lie either inside or on the polygons corresponding to the obstacles. Hereafter, all the states (joint angles) which result in collision are identified and stored. MATLAB function $'inpolygon'$ is used for this purpose.

The value function of these states which result in collision are set to 0 throughout the analysis. This simulates the obstacle avoidance behavior of the robotic arm.

C. ϵ-greedy Approach Simulation

In Q-learning, ϵ is a hyper-parameter defined by the user at the start of the algorithm. It can be a constant number between 0 and 1 or it can also be decaying as the episodes progress. For simulating this approach, in each iteration, a random number is generated between 0 and 1. If this random number is less than ϵ, then an action is chosen at random from the 9 available actions. Otherwise, the action is chosen based on the maximum Q-value.

V. Results

This section provides the results of path planning using the methods discussed in the earlier sections of this report. The end-effector's trail is shown as a black curve with arrows in each figure. It should be noted that the length of each link in these arms is 1.

A. Value Iteration

Fig. 5 shows the optimal path generated by value iteration algorithm for a 2-link planar arm. There are no obstacles in the environment in this run. The values are found to be converging after 51 iterations.

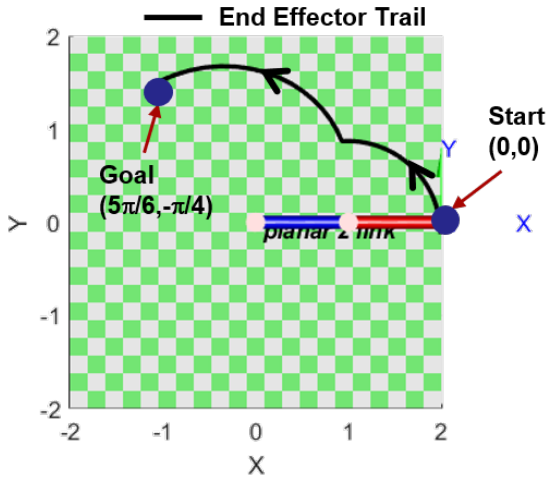

Fig. 5. 2-Link Arm Optimal Path - Value Iteration (without Obstacles)

Fig. 6 shows the optimal path generated by value iteration algorithm for a 2-link planar arm with obstacles. The values are found to be converging after 111 iterations.

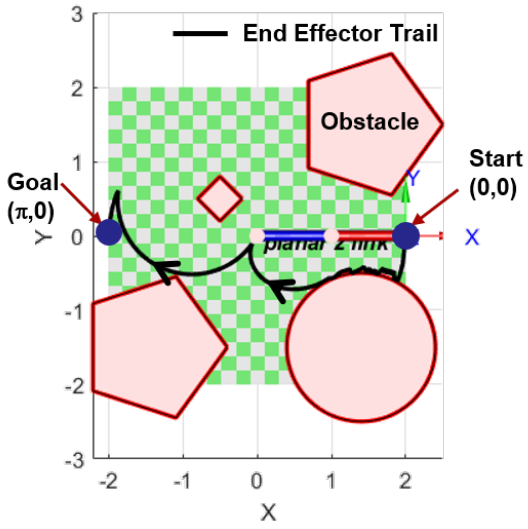

Fig. 6. 2-Link Arm Optimal Path - Value Iteration (with Obstacles)

It can be seen that when the obstacles are present, the algorithm finds the optimal solution by avoiding obstacles on its way to the goal.

Fig. 7 shows the optimal path generated by value iteration algorithm for a 3-link planar arm without obstacles. The values are found to be converging after 26 iterations. The number of iterations is less than that of a 2-link arm because the grid size is made relatively coarse due to large number of total states (around 1 million).

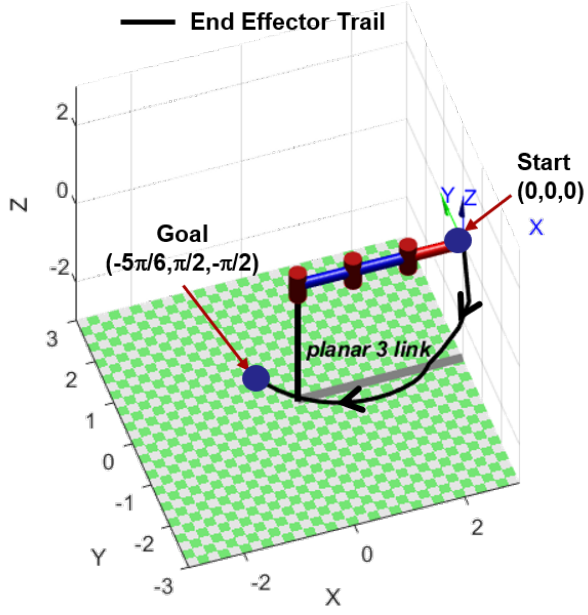

Fig. 7. 3-Link Arm Optimal Path - Value Iteration (without Obstacles)

Fig. 8 shows the optimal path generated by value iteration algorithm for a 3-link planar arm with obstacles. The values are found to be converging after 62 iterations.

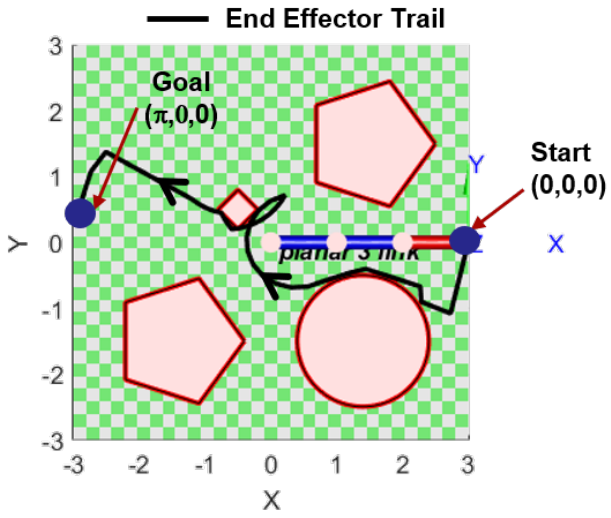

Fig. 8. 3-Link Arm Optimal Path - Value Iteration (with Obstacles)

B. Policy Iteration

Fig. 9 shows the optimal path generated by policy iteration algorithm for a 2-link planar arm with obstacles. The values are found to be converging after 76 iterations.

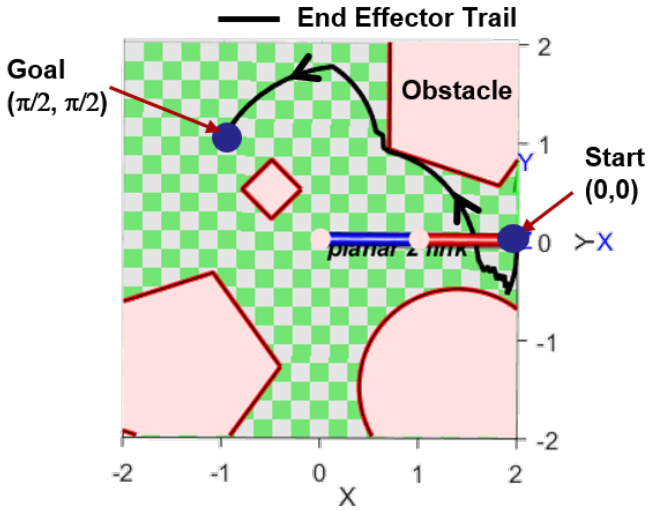

Fig. 9. 2-Link Arm Optimal Path - Policy Iteration (with Obstacles)

Fig. 10 shows the optimal path generated by policy iteration algorithm for a 3-link planar arm with obstacles. The values are found to be converging after 46 iterations.

The number of iterations performed in policy iteration is found to be relatively less than the number of iterations in value iteration.

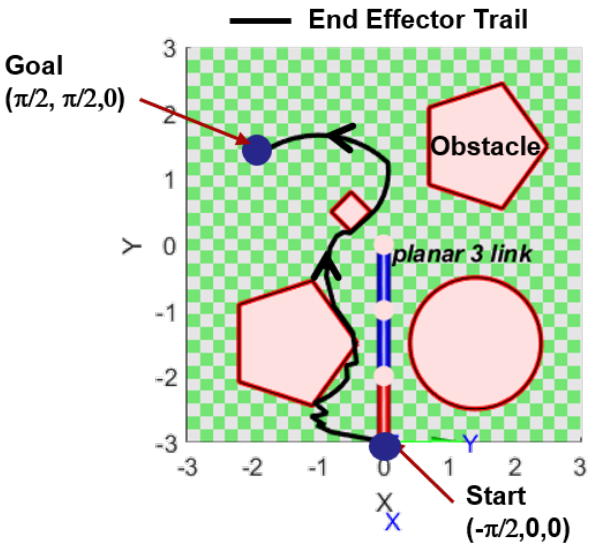

Fig. 10. 3-Link Arm Optimal Path - Policy Iteration (with Obstacles)

The policy iteration method is tried out using the same examples as value iteration. The results are found to be converging to exactly the values as before. However, the results in the previous section are provided for different start state, end state and environment just to illustrate the effectiveness of the algorithms under different conditions.

C. Q-Learning

The algorithm of Q-Learning is found not to converge to optimal policy for the current settings. Q-learning approach involves few hyper parameters (like epsilon and alpha). Also, since the number of states is huge, a very large number of episodes is required for convergence of q-values. A limit of 50,000 episodes is tried and the values didn't converge even after these episodes. Hence, the parameters are being tweaked and more iterations are being performed to improve the algorithm.

VI. Discussion of Results

Value function can be considered as a goodness measure of how good a state is for an agent. At each given state, it is a measure of the expectation of cumulative rewards over all the time steps.

It is found that high number of iterations are required for convergence as the state space is huge. The number of iterations required for convergence is also dependent on the threshold parameter if it is used to terminate the algorithm when the improvement in value function at every iteration is less than the specified value. In practice, such conditions are required for termination of value iteration algorithms. For sequential decision problems with infinite horizon, the discount factor is generally set <1 to guarantee convergence.

It is observed that the optimal state values remain the same in both value and policy iteration methods. This is consistent with the understanding that the model hasn't changed and also both the methods use Bellman's equation for solving the problem. Therefore, at convergence of V* both policy and value iteration should yield the same optimal policy π*. The value function (denoted by the colour) and the actions (arrows) should therefore be the same at convergence, since for a given MDP, there can only be one optimal policy.

The number of iterations required in policy iteration method is generally less than the number of iterations in value iteration. This is due to the fact that the policy can converge well before the values converge. The results of this project further validate that using policy iteration, faster convergence to the optimal policy can be achieved.

In Q-learning, the learning rate is chosen to be decaying (α = 1/episode). One key issue of convergence is that all state action pairs need to be continually updated, since the learned Q function directly approximates to Q*.

Notice that the epsilon parameters determines the amount of exploration and exploitation in the unknown environment. Additionally, the size of each episode in the update of the action-value function determines how quickly Q function converges to an optimal Q*. Convergence of the Q function determines how quickly the results can converge to an optimal policy. Hence fine tuning of the ϵ and α parameters is required to achieve good performance in the grid world. In order to take account of the above issues, different combinations of ϵ and α parameters are tried out. For each experiment, the number of episodes and iterations within each episode are retained and multiple iterations are performed to see the effect on the results. It is observed that a very small value of ϵ leads to more exploitation since a = $\underset{a}{\text{argmax}}\text{Q}(S_{t+1}, a)$ with probability 1-ϵ is chosen with a higher probability. Compared to that, if ϵ values are too high, then this leads to more exploration of the state space and less exploitation of the states that have already been visited. Hence, it can be seen that based on the current ϵ, α and policy chosen, the states are not explored many times which can be attributed to the lack of convergence.

VII. Major Takeaways and Future Ideas

The problem was first formulated to solve using dynamic programming. Due to the large state space and control options, the computational complexity was found to be very high. Hence, alternate methods are explored.

The idea of formulating the given problem as a sequential decision making problem proved to be very efficient. Policy and Value iteration methods are able to adapt to dynamic environments (random obstacles) easily. Q-learning approach is found to be a little trickier as it involves few hyper parameters (like epsilon and alpha). The algorithm is being improved.

In future, the current algorithm can be extended to include the notion of time in the calculations to identify the optimal trajectories of motion. Also, the effect of sensor noise can also be modeled and the effects can be studied.

VIII. Conclusions

In this report, the solution of sequential decision making problems using offline planning (Value Iteration, Policy Iteration) and online learning (Q-Learning) is discussed. The number of iterations required by the value iteration approach is found to be relative more when compared with policy iteration. Varying the starting policy in policy iteration is found to not influence the final converged optimal policy. Further, the implementation method for Q-Learning is discussed in detail. ϵ-greedy approach is used to explore the grid world. The learning rate is considered to be decaying over time. The effect of ϵ and the reference policy on the solution is discussed. Value and Policy iteration techniques are found to be efficient when the model of the problem is available to the agent. In contrast, if the model is not available, Q-learning is the better option among the three methods.

IX. Additional Files

All the MATLAB files and other supporting files used in this project are available in the following link.

https://github.com/siddharthacheruvu/EEE587-Final-Project

References

[1] Ronald A. Howard, "Dynamic Programming and Markov Processess"

[2] Peter Corke, "Robotics Toolbox for MATLAB, Release 10"

[3] Richard S.Sutton and Andrew G.Barto, "Reinforcement Learning: An Introduction, Second Edition"

[4] Stuart Russell Peter Norvig, "Artificial Intelligence - A Modern Approach, Third Edition"