Noah Ruhmer

Dynamic Programming vs. Reinforcement Learning

Bachelor Thesis

to achieve the university degree of

Bachelor of Science

Bachelor degree programme: Computer Science

submitted to

Graz University of Technology

Supervisor

Univ.-Prof. Dipl.-Ing. Dr.techn. Thomas Pock

Graz, September 2022

Affidavit

I declare that I have authored this thesis independently, that I have not used other than the declared sources/resources, and that I have explicitly indicated all material which has been quoted either literally or by content from the sources used. The text document uploaded to TUGRAZonline is identical to the present master's thesis.

Date

Signature

Abstract

In this paper, both model-based dynamic programming and model-free reinforcement learning are put into comparison. We show the necessary theory basics to work on control problems for finite discrete-time dynamic systems and how to attain an optimal policy that optimizes our objective function. Stochastic dynamic programming and Q-learning are then applied to a practical example problem that showcases the different approaches and their respective results. Our practical results show that both methods are valid approaches to solving the example.

Contents

List of Figures

1 Introduction

There are many approaches to finding optimal policies for controlling a dynamic system. However, all concepts of finding such a policy can be attributed to the 2 elementary classes: Model-Free and Model-Based algorithms. These dynamic systems describe an environment and a time-dependent objective function that we seek to control. Model-based algorithms need a complete understanding of the dynamic system, while model-free algorithms approximate the system's properties by learning a control policy.

This paper introduces the underlying theory of both approaches and shows respective methods and how they work. We analyze and compare dynamic programming to reinforcement learning on time-discrete dynamic systems. We view time as not continuous but jumping in discrete time steps in such a system.

Both methods are active fields of study, and contributions steadily augment the fundamental approach behind them.

We aim to demonstrate and analyze both methods in practice. Therefore we use the *Parking Problem* from [Ber19] and apply both algorithms to this problem to show and compare the results.

1.1 History

We will now present relevant parts of how discussed algorithms have developed throughout history. The history of these topics is quite extensive, and interested readers may want to look up Sutton and Barto [SB18] for a more detailed history.

Bellman and Richard coined the term Markov Decision Process [Bel57] in 1957 and later worked on what we now know as dynamic programming [Bel66]. The deterministic Bellman Equation 2.1 and the stochastic variant have been introduced and were used as a foundation for many upcoming enhancements in the following years. Another significant improvement on how to overcome high dimensional problems has been conceived as approximate dynamic programming by many different papers like [Si+04], [Pow07], and [Ber08].

Q-learning [Wat89] was introduced in 1989 by Watkins et al., and proof for its convergence was given in 1992 [WD92]. The basics of these works have been improved as Deep Q-learning by Gu, Shixiang, et al. [Gu+16] by using a deep convolutional neural network.

The understanding of dynamic programming and reinforcement learning has been deepened, and the relations between approaches have been compared. Busoniu, Lucian, et al. [Bus+17] compared and analyzed different algorithms as function approximators. Bertsekas [Ber19] shows an extensive comparison of both methods and similarities in the mathematical foundations.

Both topics are being actively researched, and more progress in the field of optimal control is yet to be expected.

1.2 Content

The following chapter 2 will look at the basics of optimal control problems. The class of model-free and model-based methods and several of their algorithms will be explained in more detail. In chapter 3 a simple, practical example will be solved using stochastic dynamic programming and Q-learning. This will lay out differences in approach and results depending on the method. The outlook on future work and research in this field of study and a discussion on the findings and results of this paper will be stated in the final chapter 4.

2 Background

This chapter explains the fundamentals of analyzing the dynamic programming and reinforcement learning approaches. We define a Markov Decision Process to mathematically describe and work with optimization problems.

2.1 Markov Decision Processes

A Markov Decision Process is essential to describe the system we want to find an optimal control sequence for [Bel57]. We intend to minimize the cost (Dynamic Programming 2.2) or maximize the reward (Reinforcement Learning 2.3) for our optimization problem.

These problems are discrete-time stochastic control processes meaning that the system changes in discrete steps from one state to the next. Changing from one state s to the next state s' is defined by taking action a in state s. There can be multiple valid actions in any given state, and the progression to the next step can also be stochastic. We speak of an infinite horizon problem if there is no limit on the number of successive states for a Markov Decision Process.

Lets define a Markov Decision Process as tuple: $MDP = \{S, A, P, R\}$

- S: a set of states.
- A: a set of actions.
- $P(a, s, s')$: the probability of transitioning to state s' when taking action a in state s.
- $R(s, s')$: the received reward when transitioning from state s to s'.

The Markov Property describes that a state must represent all relevant information on the system. It states that the future only depends on the current state, and all past states can be disregarded.

A special case of the Markov Decision Process would be a system in which every probability $P(a, s, s') = 1$. In this case, it is called a Deterministic Markov Decision Process as each state transition is entirely deterministic.

We can now use this system model as a basis to find an optimal control policy. This optimized policy takes actions depending on the current state and optimizes the sum of rewards until reaching an end state.

2.2 Dynamic Programming

Dynamic programming is a model-based approach associated with recursively splitting a more extensive problem into sub-problems. The sub-problems are solved independently, and their combined solutions give a solution to the whole starting problem [Bel66].

Our goal is to find the optimal policy for a process. If the process has no inert uncertainty in its system, we can progress by using deterministic dynamic programming. Otherwise, we need to account for the uncertainties, which are handled using stochastic dynamic programming [Ber12].

Dynamic programming can be used to solve problems with these properties:

- Known Markov Decision Process:
  A full description of the system model needs to be available.
- Principal of Optimality:
  Splitting the optimal solution into sub-solutions still gives the optimal solutions to the corresponding sub-problems. Therefore the problem can be split into smaller parts, which can be solved independently. This property is also called an optimal substructure. A well-known example is Dijkstra's Shortest Path, as any sub-path of the shortest path between two points is the shortest path between its endpoints.

- Overlapping sub-problems:
  A sub-problem may occur more than once within the overall problem. Dynamic programming considers this, and the sub-problem will only be solved once.

2.2.1 Deterministic Dynamic Programming

A deterministic process is a special case of the general Markov Decision Process. All $P(a, s, s') = 1$ and therefore there is only a single transition taking action a in state s leading to a single s'. Thus we can simplify the reward function to $R(a, s)$.

Deterministic Bellman equation:

$$V(s) = \max_a R(a, s) + V(s') \tag{2.1}$$

This describes that the value of a state is the sum of the next state value plus the action, which maximizes the reward between the current and next steps.

- $V(s)$: is the value function of a given state s. The value function describes the expected optimal reward from being in this state.
- $R(a, s)$: the reward received by taking action a in state s.
- $s' = t(a, s)$: t defining the deterministic transition function

The value function can then be used to find the optimal policy function $A(s)$. This function gives the optimal action a^* to take when being in state s. Furthermore, by the requirements from the Markov Decision Process, the starting states value equals this system's optimal reward.

2.2.2 Stochastic Dynamic Programming

Dynamic programming can also solve problems with an uncertainty of the next state s' [Ros14]. By extending the equation from 2.1 we can reach a

stochastic value function.

$$V(s) = \max_a \left(R(a,s) + \sum_{s'} P(a,s,s')V(s') \right) \tag{2.2}$$

- $R(a,s)$: Expected reward by taking action a in state s. This is averaged over all possible transitions from s by taking a.
- $P(a,s,s')$: Probability of transitioning from state s to s' taking action a.
- $|s'|$: Number of reached states s' by taking action a in state s.

$$R(a,s) = \frac{1}{|s'|} \sum_{s'} R(a,s,s') \cdot P(a,s,s')$$

2.3 Reinforcement Learning

Reinforcement learning is a machine learning approach in which the algorithm learns a policy to maximize the reward of a system [SB18]. This policy maps the observed environment to an action that interacts with the environment. The observed environment can be seen as the state of the Markov Decision Process.

A significant advantage of reinforcement learning is that it does not need a complete system model. It is sufficient to know states, actions, and received rewards when being or transitioning into a state. This allows reinforcement learning to work without knowing $P(a,s,s')$. It can be seen as approximated dynamic programming 2.2 as the policy indirectly learns the model and transition probabilities [Ber19].

2.3.1 Q-Learning

Q-learning is a model-free reinforcement learning algorithm. It only needs a set of all states, all actions, and the reward of taking an action while in a specific state. However, it does not need to know the probabilities of transitioning from one state to the next. It uses a Q-table which estimates

the value of taking an action in a particular state. This Q-table is a matrix containing pairs of all states and actions and depicts the value of taking a particular action in a state. The optimal policy takes the highest valued action from a state in the table [Wat89].

Q-learning's convergence has been proven, given an unlimited amount of time working on a finite Markov Decision Process [WD92]. This will converge for any initialization of the Q-table.

Training

Training is performed by selecting an action randomly or using the Q-table as the policy to choose the action. We define an exploration rate ϵ, which determines how often we select a random action, instead of using the Q-table policy to determine the action. Taking a random action is called exploration and ensures that the policy does not get stuck on a greedy local maximum but will eventually learn the global optimum.

The chosen action then gets executed, and the Q-table is updated accordingly to the received reward. This process can be repeated until we converge on a Q-table that approximates the optimal policy arbitrarily well, depending on the training time and parameters.

Update Rule

$$Q_{new}(a,s) = Q(a,s) + \alpha \cdot \left(R(a,s) + \gamma \cdot \max_{a} Q(a,s') - Q(a,s) \right) \tag{2.3}$$

This formula cannot update the final state in this table, so we set the reward of the end state (most often 0).

- α: learning rate ($0 < \alpha \leq 1$)
  This determines how much new runs influence the learned Q-Table entry. It can also be seen as learning speed, but convergence is only guaranteed for a diminishing stepsize.

- γ: discount factor ($0 \leq \gamma < 1$)
  It defines the importance of future rewards - foresight of the algorithm. Setting it to 0 will create a greedy policy, only taking the action for the best current reward into account.
  γ should not be 1 as the Q-Table values will diverge otherwise.

3 Example

We want to examine a specific problem to demonstrate differences in Dynamic Programming and Reinforcement Learning. The parking problem [Ber19] can be seen as a fully defined finite Markov Decision Process and therefore has an optimal analytical solution but also fits well to be solved by policy approximation methods such as Q-Learning 2.3.1.

3.1 Parking Problem

This problem describes a series of sequentially placed parking spaces. The driver starts at place 0 and traverses the spaces sequentially until parking at a space or reaching the last space N and parking in the garage. The driver can only observe the current parking place, free with an independent probability $p(i)$. The driver can park in a free place using the cost function $c(i)$ describing the distance the driver has to walk to his destination. Parking in the garage has a fixed cost C and no walking cost. We want the optimal policy to minimize the expected costs for a driver.

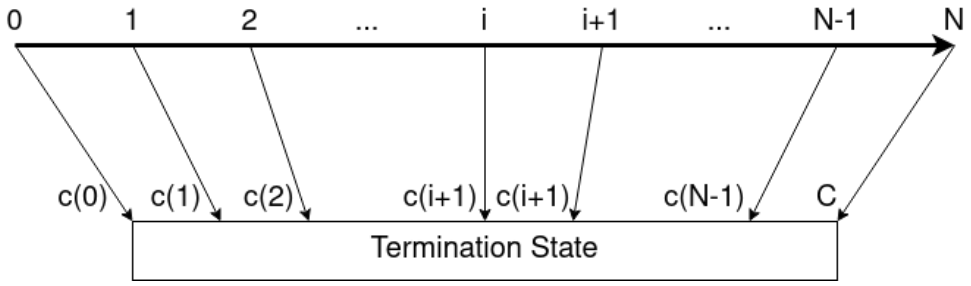

Figure 3.1: State transitions and costs of the Parking Problem

- N: Number of parking spaces
- $p(i)$: Probability that parking space i is free
- $c(i)$: Cost of the parking space i
- C: Cost of the garage

This problem gives us a total of $2N + 1$ states as for each parking place, we need one state for the place being free and one for being taken. We also need an end state indicating that the driver has parked.

This problem is stochastic as it is uncertain what the next state will be after a transition. For simplicity, we will assume a constant p, but the following solutions can also be expanded to handle a function $p(i)$.

3.2 Practical Example

We will now look at concrete values to compare different approaches to solving the stochastic Markov Decision Process with the following parameters.

The probability of a parking space being free will be constant. The cost function will increase linearly with the space index. We will set the garage cost to half of the number of spaces.

- $p(i) = p$
- $c(i) = N - i$
- $C = \frac{N}{2}$

As the cost function is linear and $c(0) > C > c(N)$, we can already expect that the best policy will be to drive up until a certain point and then park at the next free space. Therefore the policy will take the action drive for all states before this threshold and then changes to park at reaching the threshold space.

We can note from this figure 3.2 that the optimal policies and, therefore, costs are linearly dependent on the constant probability of a parking space being free. The expected result if all places are free is to park at the last possible index. If nearly every place is taken, the best policy is to try parking at space $\frac{N}{2}$ as this is at most as bad as parking in the garage.

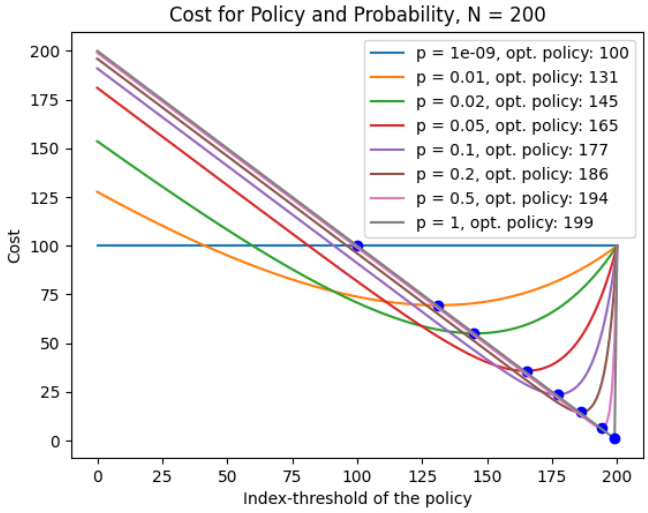

Figure 3.2: Cost of per policy and probability for the parking problem

3.2.1 Dynamic Programming Approach

As the entirety of the model is given, we can easily set up an analytical solution using Bells equation 2.2. It is crucial to notice that the value of a state does not depend on whether the previous place was free or taken. It depends only on the current index and already considers that this place is possibly taken.

The maximum from the 2 possible actions can be simplified as one already translates to the terminal state ending the recursion. We use $\gamma = 1$ as this gives the optimal solution regarding fully weighted future costs.

Recursive Value Function

$$V(N) = C = \frac{N}{2}$$
$$V(i) = p(i) \cdot c(i) + (1 - p(i)) \cdot V(i+1)$$
$$0 \leq i < N$$

Explicit Value Function

$$V(i) = C \cdot (1-p)^{N-i} + \sum_{j=0}^{N-i} p \cdot (N-i-j) \cdot (1-p)^j \tag{3.1}$$

Now that we have calculated the optimal cost using the entire model, we will approximate this solution using Q-Learning, a model-free algorithm.

3.2.2 Q-Learning Approach

Q-Learning does not need to know any probabilities or state transitions. However, it will learn to take actions that will approximate the optimal cost arbitrarily close given enough training steps.

Therefore the remaining model parameters we need to know are all states, possible actions, and the reward from transitions.

Reward Function

Q-Learning tries to maximize the reward in contrast to Dynamic Programming, which minimizes the cost. Therefore we need to transform our cost function into a corresponding reward function. We negate the cost function and garage cost 3.2 and, as Q-Learning needs positive numbers, level the reward function and garage reward to be all positive by adding N.

- $C \to R = \frac{N}{2}$
- $c(i) \to r(i) = i$

This function only rewards taking the action to park. We can instead reward transitions and give iterative rewards. It needs to hold that the sum of rewards received upon parking in any space stays equal. However, rewarding actions directly improves the performance of the Q-learning algorithm. An equally performative policy can be trained with fewer iterations than by rewarding the parking action. The following incremental reward function reduces the necessary iterations in learning to less than half of the aggregated reward function above.

- Reward from driving forward: $r(i+1) - r(i) = 1$
- Reward from transitioning to the garage: $-\frac{N}{2}$

Parameters

These parameters work well with both reward functions

- $\alpha_0 = 0.05$
- $\gamma = 0.999$
- $\epsilon_0 = 0.05$

To improve convergence speed, we use a descending learning rate α_i and exploration rate ϵ_i. With i being the current iteration and n being the number of overall iterations.

- $\alpha_i = \alpha_0 \frac{n-i+1}{n}$
- $\epsilon_i = \epsilon_0 \frac{n-i+1}{n}$

Learning Progress

For space sizes larger than 500 the learning rate can be increased while keeping the learning stable. Depending on the space size, the number of iterations also needs to be increased. For $N = 200$ and $p = 0.05$ it is enough to have 60000 learning iterations for stable results.

Using incremental rewards and diminishing learning and exploration rate, the algorithm converges at about 40000 training iterations. The plotted cost estimates are averaged from 20000 samples using the current Q-Table policy.

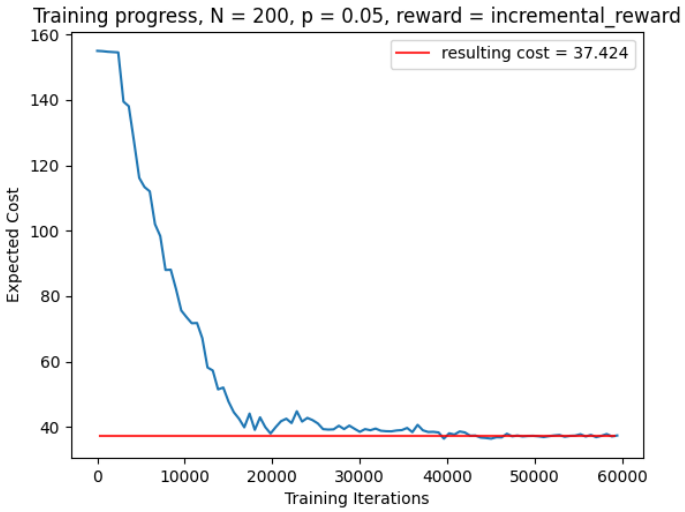

Figure 3.3: Training Progress of Q-Learning Problem, $\alpha_0 = 0.05$, $\epsilon_0 = 0.05$, $\gamma = 0.999$

Results

Q-Learning can approximate the Dynamic Programming solution quite well. The solution could be better approximated by using a better reward function, tuning the parameters, and increasing the training cycles. These results use the incremental reward function, a linearly descending learning and exploration rate.

N	Q-learning		Dynamic Programming
-	mean	standard deviation	-
50	17.005	0.263	16.366
100	26.251	0.519	25.140
200	37.145	0.989	35.764
500	53.973	1.591	51.663

The results for Q-learning are calculated from 20 learning results, each parking space randomly free with a probability of 0.05.

4 Discussion

In this paper, we discussed the basic theory for optimal control. We listed the requirements and how to define a system to describe the problem accurately. Further, we explained two highly established approaches to solving such a control problem: Dynamic programming and reinforcement learning as respective model-free and model-based methods.

We applied Stochastic Dynamic Programming and Q-Learning to an example and solved the optimal control problem with both methods. Our experimental results align with existing theory and further support that either method is valid in approaching such a problem.

Both methods differ in their advantages and disadvantages. Dynamic programming produces an optimal solution but has high prerequisites. Moreover, the solution on a particular system may not work for a slightly changed model. Model-free reinforcement learning solutions can approximate the solution arbitrarily well. They can also be more general in their solution and therefore applied to model variations. For example, the Q-learning application from the Example 3 works on any given probability distribution $p(i)$. However, the dynamic programming solution can only be applied to a constant p.

4.1 Future Work

The methods are still active research fields, and this work can be expanded upon in the future. One open question is the effects of a non-constant probability of parking spaces being free or not. A more general interest would be to apply a hybrid approach by combining dynamic programming with reinforcement learning to cancel out the disadvantages of both methods.

Bibliography

[Bel57] Richard Bellman. "A Markovian decision process." In: *Journal of mathematics and mechanics* (1957), pp. 679–684 (cit. on pp. 2, 3).

[Bel66] Richard Bellman. "Dynamic programming." In: *Science* 153.3731 (1966), pp. 34–37 (cit. on pp. 2, 4).

[Ber08] Dimitri P Bertsekas. "Approximate dynamic programming." In: (2008) (cit. on p. 2).

[Ber12] Dimitri Bertsekas. *Dynamic programming and optimal control: Volume I*. Vol. 1. Athena scientific, 2012 (cit. on p. 4).

[Ber19] Dimitri Bertsekas. *Reinforcement learning and optimal control*. Athena Scientific, 2019 (cit. on pp. 1, 2, 6, 9).

[Bus+17] Lucian Busoniu et al. *Reinforcement learning and dynamic programming using function approximators*. CRC press, 2017 (cit. on p. 2).

[Gu+16] Shixiang Gu et al. "Continuous deep q-learning with model-based acceleration." In: *International conference on machine learning*. PMLR. 2016, pp. 2829–2838 (cit. on p. 2).

[Pow07] Warren B Powell. *Approximate Dynamic Programming: Solving the curses of dimensionality*. Vol. 703. John Wiley & Sons, 2007 (cit. on p. 2).

[Ros14] Sheldon M Ross. *Introduction to stochastic dynamic programming*. Academic press, 2014 (cit. on p. 5).

[SB18] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018 (cit. on pp. 1, 6).

[Si+04] Jennie Si et al. *Handbook of learning and approximate dynamic programming*. Vol. 2. John Wiley & Sons, 2004 (cit. on p. 2).

[Wat89] Christopher John Cornish Hellaby Watkins. "Learning from delayed rewards." In: (1989) (cit. on pp. 2, 7).

[WD92] Christopher JCH Watkins and Peter Dayan. "Q-learning." In: *Machine learning* 8.3 (1992), pp. 279–292 (cit. on pp. 2, 7).